PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-259318
(43)Date of publication of application : 24.09.1999

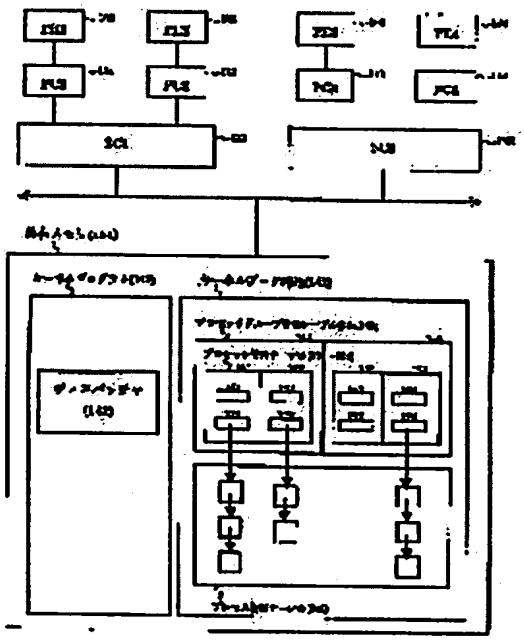
(51)Int.Cl. G06F 9/46
G06F 12/08
G06F 15/16

(21)Application number : 10-062482
(22)Date of filing : 13.03.1998

(71)Applicant : HITACHI LTD
(72)Inventor : ENKO YUTAKA
ARAI TOSHIAKI
HIGURE KOICHI
UMENO HIDENORI

(54) DISPATCH SYSTEM

(57)Abstract:
PROBLEM TO BE SOLVED: To effectively use a cache while reducing the move frequency of a sled between processors, which do not share the cache, by using group affinity even when affinity is in fail.
SOLUTION: This system is provided with means 144 and 145 for grouping and managing the processors to use the same cache memory and means 171-174 for storing the processor group to which the sled is dispatched. Thus, the processors to use the same cache memory are grouped and the sled is affiliated to plural processors in the group.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平11－259318

(43)公開日　平成11年(1999)９月24日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	9/46	３６０	Ｇ０６Ｆ	9/46	３６０Ｃ
					３６０Ｂ
	12/08			12/08	Ｈ
					Ｗ
	15/16	４３０		15/16	４３０Ｂ

審査請求　未請求　請求項の数27　ＯＬ　（全 37 頁）

(21)出願番号　特願平10－62482

(22)出願日　平成10年(1998)３月13日

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地

(72)発明者　円光　豊
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内

(72)発明者　新井　利明
神奈川県川崎市麻生区王禅寺1099番地　株式会社日立製作所システム開発研究所内

(72)発明者　日暮　浩一
神奈川県横浜市戸塚区戸塚町5030番地　株式会社日立製作所ソフトウェア開発本部内

(74)代理人　弁理士　小川　勝男

最終頁に続く

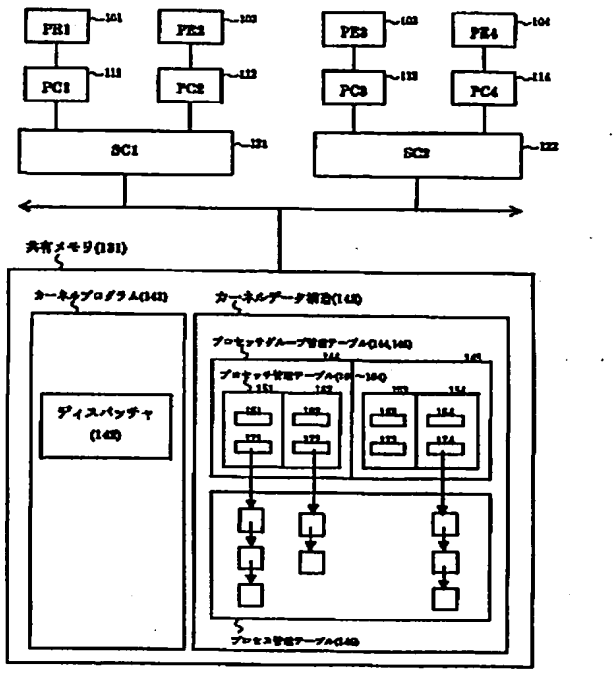
(54)【発明の名称】　ディスパッチ方式

(57)【要約】

【課題】従来のスレッド・プロセッサ間のアフィニティ付けが失敗し、スレッドをキャッシュを共有しないプロセッサへディスパッチした場合、キャッシュ間データ転送のオーバヘッドが生じる。

【構成】同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段(144,145)と、スレッドがどのプロセッサグループへディスパッチされたかを記憶する手段(171～174)を備えることにより、同一キャッシュメモリを使用するプロセッサをグループ化し、スレッドをグループ内の複数のプロセッサへアフィニティ付けする。

図１



1

【特許請求の範囲】

【請求項1】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段と、スレッドが前回ディスパッチされた際にどのプロセッサに割り付けられたかを記憶する手段を持ち、既スレッドが実行可能となった際に、既スレッドを前回割り付けたプロセッサが利用可能でなかった場合には、前回割り付けられたプロセッサが属するグループの中の利用可能なプロセッサに既スレッドを割り付けることを特徴とするディスパッチ方法。

【請求項2】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段と、スレッドが前回ディスパッチされた際にどのプロセッサに割り付けられたかを記憶する手段を持ち、プロセッサが利用可能でかつ既プロセッサが前回ディスパッチしたスレッドの中で実行可能なものがない場合には、既プロセッサが属する既グループ内のプロセッサが前回ディスパッチしたスレッドの中の実行可能なものをディスパッチすることを特徴とするディスパッチ方法。

【請求項3】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段を備え、スレッド毎に特定のプロセッサグループを指定する手段を備え、当該スレッドを前記指定のグループ内のプロセッサへのみディスパッチすることを特徴とするディスパッチ方式。

【請求項4】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段を備え、プロセス毎に特定のプロセッサグループを指定する手段を備え、当該プロセスに含まれるスレッドを前記指定のグループ内のプロセッサへのみディスパッチすることを特徴とするディスパッチ方式。

【請求項5】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段を備え、スレッド毎に特定のプロセッサグループを指定する手段を備え、システム中のいずれかのスレッドが実行可能となったとき、前記指定グループ内のプロセッサと前記指定グループ外のプロセッサが利用可能であった場合、前記指定グループ内のプロセッサへ当該スレッドをディスパッチすることを特徴とするディスパッチ方式。

【請求項6】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段を備え、スレッド毎に特定のプロセッサグループを指定する手段を備え、システム中のいずれかのプロセッサが利用可能になったとき、当該プロセッサグループを指定するスレッドと、当該プロセッサグループを指定しないスレッドが実行可能であった場合、当該プロセッサグループを指定するスレッドをディスパッチすることを特徴とするディスパッチ方式。

【請求項7】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっている

3

マルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを概プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段を備え、プロセス毎に特定のプロセッサグループを指定する手段を備え、システム中のいずれかのスレッドが実行可能となったとき、前記指定グループ内のプロセッサと前記指定グループ外のプロセッサが利用可能であった場合、前記指定グループ内のプロセッサへ当該スレッドをディスパッチすることを特徴とするディスパッチ方式。

【請求項8】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは概プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを概プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段を備え、プロセス毎に特定のプロセッサグループを指定する手段を備え、システム中のいずれかのプロセッサが利用可能になったとき、当該プロセッサが属するプロセッサグループを指定するプロセスに含まれるスレッドと、同プロセッサグループを指定しないプロセスに含まれるスレッドが実行可能であった場合、同プロセッサグループを指定するプロセスに含まれるスレッドをディスパッチすることを特徴とするディスパッチ方式。

【請求項9】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは概プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを概プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理するプロセッサグループ管理手段と、システム中に存在するスレッドをグルーピングして管理するスレッドグループ管理手段を備え、スレッドグループ毎に特定のプロセッサグループを指定する手段を備え、いづれかのスレッドが実行可能となったとき、当該スレッドの属するスレッドグループが指定するプロセッサグループ内のプロセッサと同プロセッサグループ外のプロセッサが利用可能であった場合、同プロセッサグループ内のプロセッサへ当該スレッドをディスパッチすることを特徴とするディスパッチ方式。

4

【請求項10】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは概プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを概プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理するプロセッサグループ管理手段と、システム中に存在するスレッドをグルーピングして管理するスレッドグループ管理手段を備え、スレッドグループ毎に特定のプロセッサグループを指定する手段を備え、いずれかのプロセッサが利用可能となったとき、当該プロセッサを指定するスレッドグループ内のスレッドと、当該プロセッサを指定しないスレッドグループ内のスレッドが実行可能であった場合、当該プロセッサを指定するスレッドグループ内のスレッドをディスパッチすることを特徴とするディスパッチ方式。

【請求項11】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは概プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを概プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理するプロセッサグループ管理手段と、スレッド毎に特定のプロセッサを指定する手段を備え、システム中のいずれかのスレッドが実行可能となった際に、同スレッドが指定するプロセッサが利用可能でなかった場合には、同プロセッサが属するグループ内の利用可能なプロセッサへ概スレッドを割り付けることを特徴とするディスパッチ方式。

【請求項12】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは概プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを概プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理するプロセッサグループ管理手段と、スレッド毎に特定のプロセッサを指定する手段を備え、システム中のいずれかのプロセッサが利用可能となった際に、同プロセッサを指定するスレッドの中で実行可能なものがない場合、同プロセッサ

5

が属するプロセッサグループの他のプロセッサを指定するスレッドの中で実行可能なものを割り付けることを特徴とするディスパッチ方式。

【請求項13】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理するプロセッサグループ管理手段と、プロセス毎に特定のプロセッサを指定する手段を備え、システム中のいずれかのスレッドが実行可能となった際に、当該スレッドの属するプロセスが指定するプロセッサが利用可能でなかった場合には、同プロセッサが属するグループ内の利用可能なプロセッサへ既スレッドを割り付けることを特徴とするディスパッチ方式。

【請求項14】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中のプロセスに含まれる実行可能なスレッドを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理するプロセッサグループ管理手段と、プロセス毎に特定のプロセッサを指定する手段を備え、システム中のいずれかのプロセッサが利用可能となった際に、同プロセッサを指定するプロセス内のスレッドの中で実行可能なものがない場合、同プロセッサが属するプロセッサグループの他のプロセッサを指定するプロセス内のスレッドの中で実行可能なものを割り付けることを特徴とするディスパッチ方式。

【請求項15】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中の実行可能なプロセスを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段と、プロセスが前回ディスパッチされた際にどのプロセッサに割り付けられたかを記憶する手段を持ち、既プロセスが実行可能となった際に、既プロセスを前回割り付けたプロセッサが利用可能でなかった場合には、前回割り付けられたプロセッサが属するグループの中の利用可能なプロセッサに既プロセスを割り付けることを特徴とするディスパッチ方法。

6

【請求項16】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中の実行可能なプロセスを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段と、プロセスが前回ディスパッチされた際にどのプロセッサに割り付けられたかを記憶する手段を持ち、プロセッサが利用可能でかつ既プロセッサが前回ディスパッチしたプロセスの中で実行可能なものがない場合には、既プロセッサが属する既グループ内のプロセッサが前回ディスパッチしたプロセスの中の実行可能なものをディスパッチすることを特徴とするディスパッチ方法。

【請求項17】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中の実行可能なプロセスを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段を備え、プロセス毎に特定のプロセッサグループを指定する手段を備え、当該プロセスを前記指定のグループ内のプロセッサへのみディスパッチすることを特徴とするディスパッチ方式。

【請求項18】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中の実行可能なプロセスを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段を備え、プロセス毎に特定のプロセッサグループを指定する手段を備え、システム中のいずれかのプロセスが実行可能となったとき、前記指定グループ内のプロセッサと前記指定グループ外のプロセッサが利用可能であった場合、前記指定グループ内のプロセッサへ当該プロセスを

7

ディスパッチすることを特徴とするディスパッチ方式。

【請求項19】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中の実行可能なプロセスを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段を備え、プロセス毎に特定のプロセッサグループを指定する手段を備え、システム中のいずれかのプロセッサが利用可能になったとき、当該プロセッサグループを指定するプロセスと、当該プロセッサグループを指定しないプロセスが実行可能であった場合、当該プロセッサグループを指定するプロセスをディスパッチすることを特徴とするディスパッチ方式。

【請求項20】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中の実行可能なプロセスを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理するプロセッサグループ管理手段と、システム中に存在するプロセスをグルーピングして管理するプロセスグループ管理手段を備え、プロセスグループ毎に特定のプロセッサグループを指定する手段を備え、いづれかのプロセスが実行可能となったとき、当該プロセスの属するプロセスグループが指定するプロセッサグループ内のプロセッサと同プロセッサグループ外のプロセッサが利用可能であった場合、同プロセッサグループ内のプロセッサへ当該プロセスをディスパッチすることを特徴とするディスパッチ方式。

【請求項21】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中の実行可能なプロセスを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理するプロセッサグループ管理手段と、システム中に存在するプロセスをグルーピングして管理するプロセスグループ管理手段を

8

備え、プロセスグループ毎に特定のプロセッサグループを指定する手段を備え、いずれかのプロセッサが利用可能となったとき、当該プロセッサが属するプロセッサグループを指定するプロセスグループ内のプロセスと、同プロセッサグループを指定しないプロセスグループ内のプロセスが実行可能であった場合、同プロセッサグループを指定するプロセスグループ内のプロセスをディスパッチすることを特徴とするディスパッチ方式。

【請求項22】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中の実行可能なプロセスを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理するプロセッサグループ管理手段と、プロセス毎に特定のプロセッサを指定する手段を備え、システム中のいずれかのプロセスが実行可能となった際に、同プロセスが指定するプロセッサが利用可能でなかった場合には、同プロセッサが属するグループ内の利用可能なプロセッサへ既プロセスを割り付けることを特徴とするディスパッチ方式。

【請求項23】複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが、システム中の実行可能なプロセスを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法において、

前記プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理するプロセッサグループ管理手段と、プロセス毎に特定のプロセッサを指定する手段を備え、システム中のいずれかのプロセッサが利用可能となった際に、同プロセッサを指定するプロセスの中で実行可能なものがない場合、同プロセッサが属するプロセッサグループの他のプロセッサを指定するプロセスの中で実行可能なものを割り付けることを特徴とするディスパッチ方式。

【請求項24】請求項5または7または9または11または13のディスパッチ方式において、当該指定されたプロセッサグループ内に利用可能なプロセッサが複数存在し、かつ、同プロセッサのいずれかが前回既スレッドをディスパッチしたプロセッサである場合、当該スレッドを当該プロセッサへディスパッチすることを特徴とするディスパッチ方式。

【請求項25】請求項6または8または10または12または14のディスパッチ方式において、当該プロセッサを指

定する実行可能なスレッドが複数存在し、かつ、同スレッドのいずれかが既プロセッサへ前回ディスパッチされたスレッドである場合、当該スレッドを当該プロセッサへディスパッチすることを特徴とするディスパッチ方式。

【請求項２６】請求項18または20のディスパッチ方式において、当該指定されたプロセッサグループ内に利用可能なプロセッサが複数存在し、かつ、同プロセッサのいずれかが前回既プロセスをディスパッチしたプロセッサである場合、当該プロセスを当該プロセッサへディスパッチすることを特徴とするディスパッチ方式。

【請求項２７】請求項19または21のディスパッチ方式において、当該プロセッサを指定する実行可能なスレッドが複数存在し、かつ、同プロセスのいずれかが既プロセッサへ前回ディスパッチされたプロセスである場合、当該プロセスを当該プロセッサへディスパッチすることを特徴とするディスパッチ方式。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】この発明は、複数のプロセッサと複数のキャッシュメモリで構成され、各々のキャッシュメモリは既プロセッサのなかの複数の特定のプロセッサのみが使用するデータをキャッシングするために使用される構成となっているマルチプロセッサシステムであって、ディスパッチャが実行可能なスレッドを既プロセッサ群の利用可能なプロセッサに割り付けるディスパッチ方法に関する。

【０００２】

【従来の技術】図2は、従来型の共有メモリ型マルチプロセッサシステム(SMP:Shared-memoryMulti Processor system)の例を示している。ただし、カーネルデータ構造(143)の詳細は、別図9に示している。図2のSMPの特徴は、各々のプロセッサPE1,PE2,PE3,PE4(101,102,103,104)(PEはProcessor Elementの略)が、自分自身のみがアクセス可能なキャッシュ(プライベートキャッシュPC1,PC2,PC3,PC4(111,112,113,114)(PCはPrivate Cacheの略)を有していることである。各々のキャッシュは特定のひとつのプロセッサが使用するデータをキャッシュすることができ、他のプロセッサが使用するデータをキャッシュすることはない。(PC1(111)はPE1(101)が使用するデータのみを、PC2(112)はPE2(102)が使用するデータのみを、PC3(113)はPE3(103)が使用するデータのみを、PC4(114)はPE4(104)が使用するデータのみを、キャッシュ可能である。)このようなプライベートキャッシュを有するSMPにおいて、キャッシュメモリを有効に使用する(ヒット率を向上させる)従来の方式として、スレッド(あるいはプロセス)とプロセッサとの間にアフィニティを付ける方式がある。(マルチスレッドシステムでは、プロセス(別名タスク)はプログラム実行のための論理的な単位。スレッドはプロセッサへのディスパッチの単位である。各プロセスはひとつ以上のスレッドを含む。マルチスレッドでないシステムは、プロセス中にスレッドが一つしかないマルチスレッドシステムに相当し、プロセスがプログラム実行のための単位であり、かつディスパッチの単位である。プロセスおよびスレッドについての詳細は、例えば「マルチスレッドプログラミング入門 Bil Lewis, Daniel J. Berg 共著, 岩本信一訳 アスキー出版局 ISBN 4-7561-1682-5」や、「ISO/IEC 9945-1 ANSI/IEEE Std 1003.1 POSIX 1003.1 規格書」を参照のこと。)

これは、スレッドをディスパッチするプロセッサを特定のプロセッサに限定、もしくは、利用可能なプロセッサが複数存在した場合にディスパッチ先プロセッサに優先順位をつける方式で、具体的には、例えば、カーネルが、スレッド(181～188)が前回ディスパッチされた際にどのプロセッサに割り付けられたかを記憶(本例では、図9のアフィニティ管理ポインタ(171～174)(プロセッサ管理テーブル(151～154)内にあり、プロセッサに1対1に対応している。)を先頭とするアフィニティ管理リストに各スレッドをチェーンすることによって記憶している。)し、ディスパッチャ(142)(マルチプロセス(マルチスレッドを含む)制御をおこなうOSの一部で、実行可能なスレッドをプロセッサに割り当てるプログラムのこと。詳細、および「ディスパッチ」については、「岩波・情報科学辞典」を参照のこと。)が、既スレッドが実行可能となった際に、既スレッドを前回割り付けたプロセッサに割り付ける、もしくは、プロセッサが利用可能となったとき、前回ディスパッチしたスレッドの中の実行可能なものをディスパッチする方式がある。

【０００３】このような従来のアフィニティ方式の例としては、HP-UXのmpctlシステムコールや、AIXのbindprocessシステムコールによるアフィニティ付けの例がある。これらのシステムコールを用いると、特定のスレッドを特定のプロセッサで実行するよう、OSのディスパッチャに対して推奨もしくは、強制的に指示することができる。

【０００４】また、特許における公知例としては、特開平2-33650「マルチプロセッサシステムのタスクスケジュール方式」などがある。

【０００５】次に、従来型のSMPの他の例を図3に示す。ただし、カーネルデータ構造(143)の詳細は別図9に示している。本SMPの特徴は、図の如く共有キャッシュ(121,122)を有している点にある。共有キャッシュ(SC:Shared Cache)は、システム中の複数の特定のプロセッサのみが使用するデータをキャッシュすることができ、その他のプロセッサが使用するデータをキャッシュすることはない。(SC1(121)はPE1(101)とPE2(102)が使用するデータのみを、SC2(122)はPE3(103)とPE4(104)が使用するデータのみをキャッシュ可能である。前記プライベートキャッシュとの違いは、共有キャッシュが複数

11

のプロセッサからアクセス可能な点にある。）

一般的なSMPでは、プライベートキャッシュは1次キャッシュとして、共有キャッシュは2次キャッシュとして用いられる場合が多い。（プライベートキャッシュを設けない構成もありうる。）2次キャッシュは1次キャッシュにヒットしなかったデータをキャッシングする目的で用いられ、通常、2次キャッシュへのアクセス時間はメモリよりも高速で、1次キャッシュよりも遅い。また、通常、2次キャッシュは1次キャッシュよりも大容量である。

【０００６】

【発明が解決しようとする課題】図3の如く、共有キャッシュを有するSMPにおいて、共有キャッシュを有効に使用する（ヒット率を向上させる）ために、図1で用いた従来のスレッド／プロセッサ・アフィニティ方式を採用した場合、アフィニティ付けがうまく行かなかったとき、次のような問題が生じる。

【０００７】すなわち、第1に、あるスレッドが実行可能となったとき、前回ディスパッチしたプロセッサが利用不可能（ビジー）である場合には、空いている（アイドルしている）プロセッサを利用するが、このとき、同一の共有キャッシュを共有しないプロセッサへディスパッチした場合には、それまでに共有キャッシュにキャッシュされていたデータをキャッシュ間で転送しなければならない。

【０００８】例えば、前回PE1(図3の101)にディスパッチされたスレッド(例えば図11の181)が事象待ち状態から実行可能状態になったとする。アフィニティ付けにより、スレッド(181)はプロセッサPE1(101)へディスパッチすべきだが、プロセッサPE1(101)がビジーで、プロセッサPE2,3,4(102,103,104)が利用可能(アイドル)状態だったとする。通常、共有キャッシュSC1(121)中には、スレッド(181)実行時にキャッシュされたデータが残っている。スレッド(181)がPE2(102)へディスパッチされた場合、同プロセス(181)実行にあたりPE2(102)は同キャッシュに残っているデータを高速にアクセスすることができる。しかし、PE3(102)もしくはPE4(104)へディスパッチされた場合、これらのプロセッサがSC1(121)内部に残存するデータをアクセスした場合、同データがSC1(121)からSC2(122)へキャッシュ間転送されなければならない。（公知の方式では、アフィニティ付けされていない場合、どのプロセッサへディスパッチされるかはわからない。）

第2に、システム中のいずれかのプロセッサが利用可能になったとき、キャッシュを共有しない別のプロセッサにディスパッチされたことのあるスレッドをディスパッチすると、同スレッドが、別のプロセッサの共有キャッシュ上にデータを残していた場合、当該キャッシュからデータを転送しなければならないため、ハードウェア・オーバヘッドが発生する。

12

【０００９】例えば、プロセッサPE3(図3の103)が利用可能(アイドル)状態になったとする。アフィニティ付けにより、プロセッサは前回ディスパッチしたことのあるスレッドをディスパッチすべきであるが、アフィニティ管理リスト(173)には実行可能なスレッドが無い。ここで、仮に、PE1またはPE2のアフィニティ管理リスト(171または172)から実行可能状態のスレッド（仮に184とする。）をディスパッチすると、通常、共有メモリSC1(121)中には、スレッド(184)実行時にキャッシュされたデータが残っている。このため、スレッド(184)がプロセッサPE3(103)へディスパッチされた後、SC1(121)内部に残存するデータをアクセスした場合、同データをSC1(121)からSC2(122)へキャッシュ間転送しなければならない。（公知の方式では、自分自身へアフィニティ付けされていないスレッドをディスパッチする場合、どのスレッドをディスパッチするかはわからない。）

一般に、キャッシュ／キャッシュ間の転送時間は、メモリ／キャッシュ間の転送時間よりも大きく、データ転送のためのオーバヘッドは甚大であり、CPI(Cycle Per Instruction:命令あたりの所要サイクル数)性能の低下は免れない。

【００１０】

【課題を解決するための手段】本発明では、上記の課題を解決するために、プロセッサ群の中で同一のキャッシュメモリを使用するプロセッサをグルーピングして管理する手段（プロセッサグループ管理手段）と、スレッドが前回ディスパッチされた際にどのプロセッサに割り付けられたかを記憶する手段（割付記憶手段）を有する。

【００１１】さらに別の手段としては、前記プロセッサグループ管理手段と、前記割付記憶手段を備え、プロセスまたはスレッド毎に特定のプロセッサグループを指定する手段（プロセッサグループ指定手段）を有する。

【００１２】さらに別の手段としては、前記プロセッサグループ管理手段と、前記割付記憶手段と、システム中に存在するスレッドをグルーピングして管理する手段（スレッドグループ管理手段）を備え、スレッドグループ毎にプロセッサグループ指定手段を有する。

【００１３】本発明では、OS内のカーネルプログラムの一部であるディスパッチャが、上記手段を用いて、以下の処理を実行する。（ディスパッチャが実行するアルゴリズムのフローチャートを図10～17に示す。）

ディスパッチャは、システム中のいずれかのスレッドが実行可能となった際に、概スレッドを前回割り付けたプロセッサ（前記割付記憶手段により記憶）が利用可能でなかった場合には、前回割り付けられたプロセッサが属するグループの中の利用可能なプロセッサ（前記グループ管理手段により記憶）に概スレッドを割り付ける。

【００１４】さらに、別の作用としては、ディスパッチャは、プロセッサが利用可能でかつ概プロセッサが前回ディスパッチしたスレッド（前記割付記憶手段により記

憶）の中で実行可能なものがない場合には、概プロセッサが属する概グループ内のプロセッサが前回ディスパッチしたスレッド（前記グループ管理手段により記憶）のうち実行可能なものをディスパッチする。

【００１５】さらに別の作用としては、ディスパッチャは、前記プロセッサグループ指定手段によりグループを指定するスレッドを、前記指定グループ内のプロセッサへのみディスパッチする。

【００１６】さらに別の作用としては、ディスパッチャは、前記プロセッサグループ指定手段によりグループを指定するプロセスに含まれるスレッドを、前記指定グループ内のプロセッサへのみディスパッチする。

【００１７】さらに別の作用としては、ディスパッチャは、システム中のいずれかのスレッドが実行可能となったとき、前記プロセッサグループ指定手段による指定グループ内のプロセッサと前記指定グループ外のプロセッサが利用可能であった場合、前記指定グループ内のプロセッサへ当該スレッドをディスパッチする。

【００１８】さらに別の作用としては、ディスパッチャは、システム中のいずれかのプロセッサが利用可能になったとき、前記プロセッサグループ指定手段により当該プロセッサを指定するスレッドと、当該プロセッサを指定しないスレッドが実行可能であった場合、当該プロセッサを指定するスレッドをディスパッチする。

【００１９】さらに別の作用としては、ディスパッチャは、システム中のいずれかのスレッドが実行可能となったとき、当該スレッドが属するプロセスが、前記プロセッサグループ指定手段により指定するグループ内のプロセッサと前記指定グループ外のプロセッサが利用可能であった場合、前記指定グループ内のプロセッサへ当該スレッドをディスパッチする。

【００２０】さらに別の作用としては、ディスパッチャは、システム中のいずれかのプロセッサが利用可能になったとき、前記プロセッサグループ指定手段により当該プロセッサを指定するプロセスに含まれるスレッドと、当該プロセッサを指定しないプロセスに含まれるスレッドが実行可能であった場合、当該プロセッサを指定するプロセスに含まれるスレッドをディスパッチする。

【００２１】さらに別の作用としては、ディスパッチャは、いづれかのスレッドが実行可能となったとき、当該スレッドの属するスレッドグループ（前記スレッドグループ管理手段により記憶）が前記プロセッサグループ指定手段により指定するプロセッサグループ内のプロセッサと同グループ外のプロセッサが利用可能であった場合、同グループ内のプロセッサへ当該スレッドをディスパッチする。

【００２２】さらに別の作用としては、ディスパッチャは、いずれかのプロセッサが利用可能となったとき、当該プロセッサを指定するスレッドグループ内のスレッドと、同グループ外のスレッドが実行可能であった場合、

同グループ内のスレッドをディスパッチする。

【００２３】

【発明の実施の形態】（１）実施例1

図1は、本発明の実施例の全体図である。図4は、図1のカーネルデータ構造(143)のうち、プロセッサおよびプロセス、スレッドを管理するデータ構造の詳細図である。図12は、本実施例のディスパッチャ(142)のフローチャートである。

【００２４】本実施例のSMPは図1の如く、複数のプロセッサPE1～4(101～104)と、複数のプライベートキャッシュPC1～4(111～114)と、複数の共有キャッシュSC1,SC2(121,122)と、共有メモリ(131)を備えている。なお、本実施例では4つのプロセッサ(4way)のSMPの例を示したが、実際のプロセッサ数はいくつでもよい。16ないし32以上のプロセッサを備える場合がある。

【００２５】プライベートキャッシュPC1,PC2,PC3,PC4(111～114)は、それぞれ、プロセッサPE1,PE2,PE3,PE4(101～104)が使用するデータのみをキャッシングすることができる。

【００２６】共有キャッシュSC1(121)はPE1,PE2(101,102)が使用するデータのみをキャッシングすることができ、共有キャッシュSC2(122)はPE3,PE4(103,104)が使用するデータのみをキャッシングすることができる。

【００２７】本SMPでは、プライベートキャッシュPC1,PC2,PC3,PC4は、1次キャッシュとして、共有キャッシュSC1,SC2は、2次キャッシュとして設けられている。なお、本実施例ではすべての2次キャッシュが2つのプロセッサからアクセスされる例を示したが、実際には、キャッシュが3つ以上のプロセッサにより共有される場合がある。また、各共有キャッシュのキャッシングするプロセッサの数が異なっていてもよい。（例えば、ある共有キャッシュは2つのプロセッサのデータをキャッシュし、他の共有キャッシュは4つのプロセッサのデータをキャッシュする構成が考えられる。）

さらに、共有メモリ(131)中には、カーネルプログラム(141)とカーネルデータ構造(143)が格納されている（カーネルプログラムとカーネルデータ構造は、オペレーティングシステムの一部である。）。カーネルプログラム(141)は、ディスパッチャ(142)を含んでいる。カーネルデータ構造(143)中には、プロセッサグループ管理テーブル(144,145)および、プロセス・スレッド管理テーブル(146)を含んでいる。（詳細は図4。）

プロセス・スレッド管理テーブル(146)はシステム中のすべてのスレッドの情報を保持するテーブルであり、各スレッドのスレッド構造体(181～188)を含んでいる。スレッド構造体はスレッド毎に生成され、各スレッドの情報（スレッドの実行状態（「実行中/実行可能/事象待ち/...など）、コンテクストなど）を格納している。

【００２８】本実施例では、複数のプロセッサが同一の2次キャッシュを共有している。ここでは、2次キャッ

15

シュを共有するプロセッサをメンバーとしてグループを定義し、プロセッサグループと呼ぶことにする。(図1のSMPでは、PE1とPE2がSC1を共有しており、これらが一つのプロセッサグループ(PG1とする) を形成している。また、PE3とPE4がSC2を共有しており、これらがもうひとつのプロセッサグループ(PG2とする) を形成している。)

2次キャッシュを共有するプロセッサどうしをグルーピングしてアフィニティ管理するために、カーネルデータ構造体中に、プロセッサグループ管理テーブルを備え、さらに、同テーブル中にプロセッサ管理テーブルを備える。

【００２９】プロセッサグループ管理テーブル(144,145)は、プロセッサグループに1対1に対応して存在している。(図1では、プロセッサグループ管理テーブルGT1(144)がプロセッサグループ(メンバPE1,PE2)に対応しており、プロセッサグループ管理テーブルGT2(145)がプロセッサグループPG2(メンバPE2,PE3)に対応している。)

プロセッサ管理テーブル(151,152,153,154)はプロセッサグループ管理テーブル(144,145)中にあり、プロセッサに1対1に対応して存在している。(図1では、PT1がPE1に、PT2がPE2に、...対応している。)

各プロセッサ管理テーブルとそれを包含するプロセッサグループ管理テーブルとの関係は、対応するプロセッサとそれに含まれるプロセッサグループとの関係と一致している。プロセッサ管理テーブルは、それに対応するプロセッサが所属するグループに対応するグループアフィニティ管理テーブルに格納されている。(PT1とPT2はGT1に、PT3とPT4はGT2に格納されている。)

各々のプロセッサ管理テーブル(151～154)は、スレッド構造体(181～188)のうち、前回概プロセッサへディスパッチしたものをリスト管理している。(本例では、プロセッサアフィニティ管理ポインタ(171)を先頭とするチェーンをたどることにより、該当するスレッドがすべて検索できるデータ構造となっている。ポインタチェーンによるデータ構造(リスト構造) は、一般的なプログラミングのモデルとして使用されるので、文献「アルゴリズム＋データ構造＝プログラム」(N.Wirth著)等の文献を参照されたい。) システム中のすべてのスレッド構造体はいづれかのプロセッサアフィニティ管理リストに属している。本実施例では、これにより、スレッドが前回ディスパッチされた際にどのプロセッサに割り付けられたかを記憶している。

【００３０】次に、本実施例のディスパッチャの動作を説明する。システム中の各プロセッサは、レベル2キャッシュに合せてグルーピングされており、ディスパッチャは、レベル2キャッシュを活用するために、プロセッサアフィニティが失敗した場合にはグループアフィニティを使用する。以下にその詳細を説明する。

16

【００３１】まず、スレッド生成時、当該スレッドのスレッド構造体は、生成されたプロセッサのプロセッサ管理テーブルのアフィニティ管理リストのメンバーとなる(アフィニティ管理ポインタを先頭とするポインタのチェーンへリンクされる。)。

【００３２】図12は、本発明の第1の実施例のディスパッチャ(142)のフローチャートである。

【００３３】本フローチャートに示されたアルゴリズムは、システム中のあるスレッドが実行可能状態になったとき、システム中のいずれかのプロセッサにより実行される。

【００３４】まず、ディスパッチャは、実行可能になったスレッドを前回割り付けたプロセッサが現在利用可能(アイドル状態) かどうか検査する(1201)。(各プロセッサの状態(利用可能かどうか) の検査は、各プロセッサ管理テーブル中のプロセッサ状態を表す変数(161～164)(これは、一般的なオペレーティングシステムが有しており、カーネルがスレッドディスパッチを実行するたびに更新する。プロセッサ状態を表す変数は、共有メモリ中にあり、どのプロセッサも、すべてのプロセッサ状態を検査することが可能である。) を参照することによって行なう。)

もしも、同プロセッサが利用可能ならば、同プロセッサへディスパッチする。(1202)

もしも、同プロセッサが利用可能でないならば、次に、前回ディスパッチしたプロセッサと同一グループ内に利用可能なプロセッサがあるかどうか検査する。(1203)(同一グループ内のプロセッサは、次のように検索する。例えば、PE1(101)と同一グループ内のプロセッサは、PE1(101)に対応するプロセッサ管理テーブルPT1(147)が格納されているグループ管理テーブルGT1(144)に格納されている他のプロセッサ管理テーブル、ここでは、PT2(152)に対応するプロセッサPE2(102)である。PE2プロセッサの状態(利用可能かどうか) の検査はプロセッサ状態変数(162)によっておこなう。)

もしも、同一グループ内に利用可能なプロセッサがあれば、同プロセッサに当該スレッドをディスパッチする。(1204)(ディスパッチの際、ディスパッチャは、プロセッサアフィニティ管理リスト(171～174)の更新を行なう。すなわち、当該スレッド構造体を旧来のプロセッサアフィニティ管理ポインタのチェーンから外し、ディスパッチ先のプロセッサアフィニティ管理ポインタのチェーンへつなぐ。リスト操作のアルゴリズムは前記文献等を参照のこと。)

もしも、同一グループ内に利用可能なプロセッサがなければ、グループ外の利用可能なプロセッサが存在するかどうか検査する。(1205)(グループ外のプロセッサについては、自分自身を除く全てのプロセッサグループ管理テーブル内の、全てのプロセッサ管理テーブルのプロセッサ状態記憶変数を検査すればよい。)

もしも、グループ外の利用可能なプロセッサが存在すれば、同プロセッサにディスパッチし(406)、存在しなければ、システム中のいづれかのプロセッサが利用可能になるのを待つ。(407)

(2) 実施例2

次に、本発明の別の実施例を示す。本実施例の全体図は図1、カーネルデータ構造の詳細図は図4であり、ディスパッチャのアルゴリズム以外は、実施例1と同一である。

【0035】本実施例のディスパッチャ(142)のアルゴリズムを示すフローチャートを図13に示す。

【0036】本フローチャートに示されたアルゴリズムは、システム中のプロセッサのいずれかが利用可能(アイドル状態)になったとき、同プロセッサにより実行される。

【0037】まず、前回同プロセッサへディスパッチされたスレッドのうち、実行可能なものが存在するかどうか検査する。(1301)(実行可能なスレッドの検索は、当該プロセッサに対応するプロセッサ管理テーブル(151~154)中のアフィニティ管理ポインタ(171~174)からアフィニティ管理リストをたどり、各スレッド構造体(181~188)中に格納されているスレッド状態変数(図4の401)が「READY(実行可能)」であるものを検索する。以下同じ。)

もしも、そのようなスレッドが存在すれば、同スレッドをディスパッチする(1302)。

【0038】もしも、そのようなスレッドが存在しなければ、同プロセッサと同一グループ内のプロセッサにアフィニティ付けされた実行可能なスレッドが存在するかどうか検査する。(1303)(これは、次のように行なう。例えば、プロセッサPE1(101)と同一グループのプロセッサにアフィニティ付けされたスレッドは、PE1と同一のプロセッサグループ管理テーブルGT1(144)内のプロセッサ管理テーブルPT2(152)内のアフィニティ管理ポインタ(172)にリンクされているスレッド構造体(184,185)のスレッド状態変数(404,405)を検査する。)

もしも、そのようなスレッドが存在すれば、同スレッドをディスパッチする(1304)

もしも、そのようなスレッドが存在しなければ、他のグループに実行可能なスレッドが存在するかどうか検査する。(1305)

もしも、グループ外に実行可能なスレッドが存在すれば、同スレッドをディスパッチし(1306)、存在しなければシステム中のいずれかのスレッドが実行可能な状態になるのを待つ(アイドル状態で待つ)。(1307)

なお、実施例1,2のアフィニティ方式を実現するためには、カーネル管理データ構造(143)が図5のデータ構造であってもよい。(図1のカーネルデータ構造(143)中のプロセッサグループ管理テーブル(144)が、プロセッサ情報テーブル(501)に置き換わる。)

図5のプロセッサ情報テーブル(501)の各行の項目は、プロセッサ番号(511),グループ内の次のプロセッサ番号(512)、プロセッサ状態(513)、アフィニティ管理ポインタ(514)、その他である。同テーブルの列数はプロセッサ数であり、各プロセッサに関する各項目毎の情報が同一列に格納される。

【0039】「プロセッサ番号」(511)の行には、システム中のプロセッサのID(通し番号)が格納される。「グループ内の次のプロセッサ番号」(512)の行には、次にようにプロセッサIDを格納する。すなわち、同一グループ(前述)に属するプロセッサをすべて集めて順にならべ、各プロセッサの次の(ただし、最後のプロセッサには最初の)順番のプロセッサの番号を格納する。例えば、図7のように、プロセッサ3と同じグループのプロセッサを順にならべると、(3,4)であり、プロセッサ3の次のプロセッサは4。プロセッサ3の次のプロセッサは4である。こうしておけば、各々のプロセッサについて、同一グループに属するプロセッサを検索するには、同プロセッサに対応する列の「グループ内の次のプロセッサ番号」(512)の項目を(自分自身のプロセッサ番号が出現するまで)順にたどればよい。

【0040】「アフィニティ管理ポインタ」(514)の行には、前回概プロセッサへディスパッチしたスレッドのスレッド構造体リストの先頭アドレスが格納されている。(図1のアフィニティ管理ポインタ(171~174)と同様の内容が格納される。)

第2の実施例のプロセッサ情報テーブル(501)を用いた場合のディスパッチャの動作は、第1の実施例と同様である。(図10,11のフローチャート参照。)

ただし、同一グループ内の利用可能なプロセッサの検索は次のように行う。まず、各々のプロセッサに対応する列の「グループ内の次のプロセッサ番号」(512)の項目をみる。そして、同「プロセッサ番号」(511)の列の「プロセッサ状態」(513)の行を検査し「IDLE」であれば、当該プロセッサは利用可能である。もしも「RUN」であれば、当該プロセッサは利用できないので、次に、当該プロセッサの「グループ内の次のプロセッサ番号」(512)の項目をみる。そして、同番号の「プロセッサ番号」(511)の列の「プロセッサ状態」(513)の行を検査する。これを、「グループ内の次のプロセッサ番号」(512)に自分自身のプロセッサ番号が出現するまで繰り返す。

【0041】また、同一グループ内の実行可能なスレッドの検索は次のように行う。上記の方法で、検索した同一グループのプロセッサへアフィニティ付けされたスレッドは、当該プロセッサの「プロセッサ番号」(511)の列の「アフィニティ管理ポインタ」(514)を先頭としたリストをたどることによって、検索できる。同リストのメンバーのスレッド構造体中にあるの「スレッド状態変数」(401)を参照することによって、同スレッドが実行

可能か判別できる。

【００４２】（３）実施例3

次に、本発明の別の実施例を示す。本実施例の全体図は図1であるが、カーネルデータ構造の詳細が図6に示すデータ構造に置き換わっている。ディスパッチャのアルゴリズムを図14のフローチャートによって示す。

【００４３】本実施例のプロセッサグループ管理テーブル(144,145)は、実施例1と同様、各プロセッサに対応したプロセッサ管理テーブル(151～154)を格納している。プロセッサ管理テーブル(151～154)は、プロセッサ状態記憶変数(161～164)を含んでおり、各プロセッサへスレッドをディスパッチ可能かどうかを表わしている。

【００４４】プロセス管理テーブル(146)中には、システム中に存在するすべてのプロセスのプロセス構造体(601～603)が格納されており、各プロセス構造体中には、当該プロセス中に生成されたすべてのスレッドのスレッド構造体(611～618)が格納されている。

【００４５】各スレッド構造体中(611～618)には、スレッド実行状態変数(631～638)とプロセッサグループ指定変数(631～638)が格納されている。スレッド実行状態変数(631～638)には、各スレッドの実行状態(実行可能/事象待ち/...)が格納されている。プロセッサグループ指定変数(631～638)には、特定のプロセッサグループを指定するデータ(プロセッサグループ管理テーブルへのポインタもしくはプロセッサグループID、ここではプロセッサグループIDとする。)が格納される。

【００４６】プロセッサグループ指定変数(631～638)へのグループIDのセット方法としては、OSが同変数設定のためのシステムコールを用意し、システム中で動作するスレッドが実行するプログラムもしくは、ユーザコマンドから、同システムコールを介してカーネルプログラムに設定させる方法が考えられる。

【００４７】本実施例では、上記プロセッサグループ指定変数(631～638)によって、スレッド毎に特定のプロセッサグループを指定する。

【００４８】次に、本実施例のディスパッチャ(142)のアルゴリズムを図14のフローチャートに示す。本フローチャートに示されたアルゴリズムは、システム中のいずれかのスレッドが実行可能状態になったとき、当該スレッドが指定したプロセッサグループ内のプロセッサにより実行される。(指定したグループ以外のプロセッサは実行しない。)

まず、実行可能になった当該スレッドが指定するプロセッサグループ内のいずれかのプロセッサが利用可能かどうか検査する(1401)。これは、図6のカーネルデータ構造において次のように行う。すなわち、例えば、スレッド(614)が実行可能になったとする。同スレッドのプロセッサグループ(PG)指定変数(634)を参照し(ここでは「１」。)、当該プロセッサグループ管理テーブルPG1(144)中のプロセッサ管理テーブルPT1(151)とPT2(152)のプロセッサ状態変数(161と162)を参照する。これらのいずれかが「利用可能(IDLE)」を表わしていれば、当該プロセッサへディスパッチを行う(1402)。(この場合、前回ディスパッチしたプロセッサへディスパッチすることが好ましい。前回ディスパッチしたプロセッサを選択する手段とアルゴリズムは実施例1、２と同様。)いずれも「利用不可(BUSY)」であった場合には、当該スレッドは実行可能状態のまま、指定グループ内のいずれかのプロセッサが利用可能になるのを待って休眠(sleep)する(1403)。(グループ外のプロセッサが利用可能であっても、それらへはディスパッチしない。)

上記方式によって、当該スレッドを、スレッドが指定するグループ内のプロセッサへのみディスパッチすることが出来る。

【００４９】（４）実施例4

次に、本発明の別の実施例を示す。本実施例の全体図は図1であるが、カーネルデータ構造の詳細が図7に示すデータ構造に置き換わっている。ディスパッチャのアルゴリズムは図15のフローチャートである。

【００５０】本実施例のプロセッサグループ管理テーブル(144,145)は、実施例1と同様、各プロセッサに対応したプロセッサ管理テーブル(151～154)を格納している。プロセッサ管理テーブル(151～154)は、プロセッサ状態記憶変数(161～164)を含んでおり、各プロセッサへスレッドをディスパッチ可能かどうかを表わしている。

【００５１】プロセス管理テーブル(146)中には、システム中に存在するすべてのプロセスのプロセス構造体(601～603)が格納されており、各プロセス構造体中には、当該プロセス中に生成されたすべてのスレッドのスレッド構造体(611～618)と、プロセッサグループ指定変数(701～703)が格納されている。プロセッサグループ指定変数(701～703)には、特定のプロセッサグループを指定するデータ(プロセッサグループ管理テーブルへのポインタもしくはプロセッサグループID、ここではプロセッサグループIDとする。)が格納される。

【００５２】各スレッド構造体中(611～618)には、スレッド実行状態変数(631～638)が格納されている。スレッド実行状態変数(631～638)には、各スレッドの実行状態(実行可能(READY)/事象待ち(WAIT)/実行中(RUN)/...のいずれか)が格納されている。

【００５３】プロセッサグループ指定変数(701～703)へのグループIDのセット方法としては、OSが同変数設定のためのシステムコールを用意し、システム中で動作するスレッドが実行するプログラムもしくは、ユーザコマンドから、同システムコールを介してカーネルプログラムに設定させる方法が考えられる。

【００５４】本実施例では、上記プロセッサグループ指定変数(701～703)によって、プロセス毎に特定のプロセッサグループを指定する。

【００５５】次に、本実施例のディスパッチャ(142)の

アルゴリズムを図15のフローチャートに示す。本フローチャートに示されたアルゴリズムは、システム中のいずれかのスレッドが実行可能状態になったとき、当該スレッドが指定したプロセッサグループ内のプロセッサにより実行される。(指定したグループ以外のプロセッサは実行しない。)

まず、実行可能になった当該スレッドの属するプロセスが指定するプロセッサグループ内のいずれかのプロセッサが利用可能かどうか検査する(1501)。これは、図7のカーネルデータ構造において次のように行う。すなわち、例えば、スレッド(614)が実行可能になったとする。同スレッドが所属するプロセス(602)のプロセッサグループ(PG)指定変数(702)を参照し(ここでは「1」。)、当該プロセッサグループ管理テーブルPG1(144)中のプロセッサ管理テーブルPT1(151)とPT2(152)のプロセッサ状態変数(161と162)を参照する。これらのいずれかが「利用可能(IDLE)」を表わしていれば、当該プロセッサへディスパッチを行う(1502)。(複数のプロセッサが利用可能であった場合には前回ディスパッチしたプロセッサへディスパッチするのが望ましい。)いずれも「利用不可(BUSY)」であった場合には、当該スレッドは実行可能状態のまま、指定グループ内のいずれかのプロセッサが利用可能になるのを待って休眠(sleep)する(1503)。(グループ外のプロセッサが利用可能であっても、それらへはディスパッチしない。)

(5)実施例5

次に、本発明の別の実施例を示す。本実施例の全体図は図1であるが、カーネルデータ構造の詳細が図6に示すデータ構造に置き換わっている。ディスパッチャのアルゴリズムは図16のフローチャートである。

【0056】本実施例のカーネルデータ構造については、実施例3の説明を参照されたい。

【0057】本実施例のディスパッチャのアルゴリズム(図13)は、システム中のいずれかのスレッドが実行可能状態になったとき、システム中のいずれかのプロセッサにより実行される。

【0058】まず、実行可能になった当該スレッドが指定するプロセッサグループ内のいずれかのプロセッサが利用可能かどうか検査する(1601)。これは、図6のカーネルデータ構造において次のように行う。すなわち、例えば、スレッド(614)が実行可能になったとする。同スレッドのプロセッサグループ(PG)指定変数(634)を参照し(ここでは「1」。)、当該プロセッサグループ管理テーブルPG1(144)中のプロセッサ管理テーブルPT1(151)とPT2(152)のプロセッサ状態変数(161と162)を参照する。

【0059】これらのいずれかが「利用可能(IDLE)」を表わしていれば、当該プロセッサへディスパッチを行う。(複数のプロセッサが利用可能であった場合には、前回ディスパッチしたプロセッサへディスパッチするのが好ましい。本手段とアルゴリズムは前記実施例と同様。)(1602)

いずれも「利用不可(BUSY)」であった場合には、指定プロセッサグループ外のいずれかのプロセッサが利用可能かどうか検査する(1603)。これは、図6のカーネルデータ構造において次のように行う。すなわち、プロセッサグループ管理テーブルGT1(144)以外のすべてのプロセッサグループ管理テーブル(ここでは、GT2)に含まれるすべてのプロセッサ管理テーブル(ここでは、PT3とPT4)中のプロセッサ状態記憶変数(ここでは、163,164)を参照する。

【0060】これらのいずれかが「利用可能(IDLE)」を表わしていれば、当該プロセッサへディスパッチを行う。(1604)

いずれも「利用不可(BUSY)」であった場合には、当該スレッドは実行可能状態のまま、システム中のいずれかのプロセッサが利用可能になるのを待って休眠(sleep)する(1605)。

【0061】(6)実施例6

次に、本発明の別の実施例を示す。本実施例の全体図は図1であるが、カーネルデータ構造の詳細が図6に示すデータ構造に置き換わっている。ディスパッチャのアルゴリズムは図17のフローチャートである。

【0062】本実施例のカーネルデータ構造については、実施例3の説明を参照されたい。

【0063】本実施例のディスパッチャのアルゴリズム(図17)は、システム中のいずれかのプロセッサが利用可能になったとき、同プロセッサにより実行される。

【0064】まず、利用可能になったプロセッサを含むプロセッサグループを指定するスレッドが存在するかどうか検査する(1701)。これは、図6のカーネルデータ構造において次のように行う。すなわち、例えば、プロセッサPE2を指定するスレッドを検索するには、プロセス管理テーブル(146)中のすべてのプロセス構造体(601～603)の全スレッド構造体(611～618)中のプロセッサグループ指定変数(631～638)を参照し、同変数がプロセッサPE2を指定するスレッドのスレッド実行状態変数(621～628)を参照する。

【0065】これらのいずれかが「実行可能(READY)」であれば、それらのスレッドの一つを同プロセッサへディスパッチする(1402)。

【0066】これらのいずれもが「実行可能(READY)」でなければ、システム中のすべてのスレッドのうち、実行可能なものが存在するかどうか検査する(1703)。これは、図6のカーネルデータ構造において次のように行う。すなわち、プロセス管理テーブル(146)中のすべてのプロセス構造体(601～603)の全スレッド構造体(611～618)中のプロセッサグループ指定変数(631～638)を参照し、同変数がプロセッサPE2を指定しないスレッド(PE2を指定するスレッドは検査済みなので。)のスレッド実

行状態変数(621～628)を参照する。

【００６７】これらのいずれかが「実行可能(READY)」であれば、それらのスレッドの一つを同プロセッサへディスパッチする(1704)。

【００６８】これらのいずれもが「実行可能(READY)」でなければ、当該プロセッサは、プロセッサ管理テーブルのプロセッサ状態変数を「利用可能(idle)」にして、システム中のいずれかのスレッドが実行可能になるのを待つ(1705)。

【００６９】（７）実施例7

次に、本発明の別の実施例を示す。本実施例の全体図は図1であるが、カーネルデータ構造の詳細が図7に示すデータ構造に置き換わっている。ディスパッチャのアルゴリズムは図16のフローチャートである。

【００７０】本実施例のカーネルデータ構造については、実施例4の説明を参照されたい。

【００７１】本実施例のディスパッチャのアルゴリズム(図16)は、システム中のいずれかのスレッドが実行可能状態になったとき、システム中のいずれかのプロセッサにより実行される。

【００７２】まず、実行可能になった当該スレッドが含まれるプロセスが指定するプロセッサグループ内のいずれかのプロセッサが利用可能かどうか検査する(1601)。これは、図6のカーネルデータ構造において次のように行う。すなわち、例えば、スレッド(614)が実行可能になったとする。同スレッドのスレッド構造体(614)が含まれるプロセス構造体(602)のプロセッサグループ(PG)指定変数(702)を参照し（ここでは「１」。）、当該プロセッサグループ管理テーブルPG1(144)中のプロセッサ管理テーブルPT1(151)とPT2(152)のプロセッサ状態変数(161と162)を参照する。

【００７３】これらのいずれかが「利用可能(IDLE)」を表わしていれば、当該プロセッサへディスパッチを行う。(1602)

いずれも「利用不可(BUSY)」であった場合には、指定プロセッサグループ外のいずれかのプロセッサが利用可能かどうか検査する(1603)。これは、図6のカーネルデータ構造において次のように行う。すなわち、プロセッサグループ管理テーブルGT1(144)以外のすべてのプロセッサグループ管理テーブル（ここでは、GT2）に含まれるすべてのプロセッサ管理テーブル（ここでは、PT3とPT4）中のプロセッサ状態記憶変数（ここでは、163,164）を参照する。

【００７４】これらのいずれかが「利用可能(IDLE)」を表わしていれば、当該プロセッサへディスパッチを行う。(1604)

いずれも「利用不可(BUSY)」であった場合には、当該スレッドは実行可能状態のまま、システム中のいずれかのプロセッサが利用可能になるのを待って休眠(sleep)する(1605)。

【００７５】（８）実施例8

次に、本発明の別の実施例を示す。本実施例の全体図は図1であるが、カーネルデータ構造の詳細が図7に示すデータ構造に置き換わっている。ディスパッチャのアルゴリズムは図18のフローチャートである。

【００７６】本実施例のカーネルデータ構造については、実施例4の説明を参照されたい。

【００７７】本実施例のディスパッチャのアルゴリズム(図18)は、システム中のいずれかのプロセッサが利用可能になったとき、同プロセッサにより実行される。

【００７８】まず、利用可能になったプロセッサを含むプロセッサグループを指定するプロセス中に実行可能なスレッドが存在するかどうか検査する(1801)。これは、図7のカーネルデータ構造において次のように行う。すなわち、例えば、プロセッサPE2を指定するプロセスを検索するには、プロセス管理テーブル(146)中のすべてのプロセス構造体(601～603)のプロセッサグループ指定変数(701～703)を参照し、同変数がプロセッサPE2を指定するプロセス(図7では、602)のスレッド構造体のスレッド実行状態変数(614,615)を参照する。

【００７９】これらのいずれかが「実行可能(READY)」であれば、それらのスレッドの一つを同プロセッサへディスパッチする(1802)。

【００８０】これらのいずれもが「実行可能(READY)」でなければ、システム中のすべてのスレッドのうち、実行可能なものが存在するかどうか検査する(1803)。これは、図7のカーネルデータ構造において次のように行う。すなわち、プロセス管理テーブル(146)中のすべてのプロセス構造体(601～603)の全スレッド構造体(611～618)中のプロセッサグループ指定変数(631～638)を参照し、同変数がプロセッサPE2を指定しないスレッド（PE2を指定するスレッドは検査済みなので。）のスレッド実行状態変数(621～628)を参照する。

【００８１】これらのいずれかが「実行可能(READY)」であれば、それらのスレッドの一つを同プロセッサへディスパッチする(1804)。

【００８２】これらのいずれもが「実行可能(READY)」でなければ、当該プロセッサは、プロセッサ管理テーブルのプロセッサ状態変数を「利用可能(idle)」にして、システム中のいずれかのスレッドが実行可能になるのを待つ(1805)。

【００８３】（９）実施例9

次に、本発明の別の実施例を示す。本実施例の全体図は図1であるが、カーネルデータ構造の詳細が図8に示すデータ構造に置き換わっている。ディスパッチャのアルゴリズムは図19のフローチャートである。

【００８４】本実施例のプロセッサグループ管理テーブル(144,145)は、実施例1と同様、各プロセッサに対応したプロセッサ管理テーブル(151～154)を格納している。プロセッサ管理テーブル(151～154)は、プロセッサ

状態変数(161～164)を含んでおり、各プロセッサへスレッドをディスパッチ可能かどうかを表わしている。

【００８５】プロセス管理テーブル(146)中には、システム中に存在するすべてのプロセスのプロセス構造体(601～603)と、スレッドグループ管理構造体(801～803)が格納されている。各プロセス構造体中(601～603)には、当該プロセス中に生成されたすべてのスレッドのスレッド構造体(611～618)が格納されている。スレッド構造体(611～618)にはスレッド実行状態変数(621～628)とスレッドグループリンクポインタ(831～838)が格納されている。

【００８６】スレッドグループ管理構造体(801～803)には、プロセッサグループ指定変数(811～813)とスレッドグループ管理ポインタ(821～823)が格納されている。スレッドグループ管理ポインタ(821～823)を先頭とするポインタチェーンリストには、0個～複数個のスレッド構造体がスレッドグループリンクポインタ(831～838)によってチェーンリンクされており、スレッドグループを作っている。各スレッドグループに属する0個～複数個のスレッドは、グループに対応する各プロセッサグループ指定変数(811～813)により特定のプロセッサグループを指定している。(図8では、例えば、スレッド611,612,614が同一のスレッドメンバーに属しており、これらは、プロセッサグループ1を指定(811による)している。)

スレッドグループの生成(スレッドグループ管理構造体の追加)、各スレッドグループへのスレッドの追加、削除、プロセッサグループ指定変数(811～813)へのグループIDのセット方法としては、OSが同実施のためのシステムコール・インターフェースを用意し、システム中で動作するスレッドが実行するプログラムもしくは、ユーザコマンドから、同システムコールを介してカーネルプログラムに実施させる方法が考えられる。

【００８７】各スレッド構造体中(611～618)には、スレッド実行状態変数(631～638)が格納されている。スレッド実行状態変数(631～638)には、各スレッドの実行状態(実行可能(READY)/事象待ち(WAIT)/実行中(RUN)/...のいずれか)が格納されている。

【００８８】本実施例のディスパッチャのアルゴリズム(図19)は、システム中のいずれかのスレッドが実行可能状態になったとき、システム中のいずれかのプロセッサにより実行される。

【００８９】まず、実行可能になった当該スレッドが含まれるグループが指定するプロセッサグループ内のいずれかのプロセッサが利用可能かどうか検査する(1901)。これは、図8のカーネルデータ構造において次のように行う。すなわち、例えば、スレッド(614)が実行可能になったとする。同スレッドのスレッド構造体(614)が含まれるリスト構造体のスレッドグループ管理構造体(801)のプロセッサグループ(PG)指定変数(811)を参照し

(ここでは「１」。)、当該プロセッサグループ管理テーブルPG1(144)中のプロセッサ管理テーブルPT1(151)とPT2(152)のプロセッサ状態変数(161と162)を参照する。

【００９０】これらのいずれかが「利用可能(IDLE)」を表わしていれば、当該プロセッサへディスパッチを行う。(1902)

いずれも「利用不可(BUSY)」であった場合には、指定プロセッサグループ外のいずれかのプロセッサが利用可能かどうか検査する(1903)。これは、図8のカーネルデータ構造において次のように行う。すなわち、プロセッサグループ管理テーブルGT1(144)以外のすべてのプロセッサグループ管理テーブル(ここでは、GT2)に含まれるすべてのプロセッサ管理テーブル(ここでは、PT3とPT4)中のプロセッサ状態記憶変数(ここでは、163,164)を参照する。

【００９１】これらのいずれかが「利用可能(IDLE)」を表わしていれば、当該プロセッサへディスパッチを行う。(1904)

いずれも「利用不可(BUSY)」であった場合には、当該スレッドは実行可能状態のまま、システム中のいずれかのプロセッサが利用可能になるのを待って休眠(sleep)する(1905)。

【００９２】(１０)実施例10

次に、本発明の別の実施例を示す。本実施例の全体図は図1であるが、カーネルデータ構造の詳細が図8に示すデータ構造に置き換わっている。ディスパッチャのアルゴリズムは図20のフローチャートである。

【００９３】本実施例のカーネルデータ構造については、実施例9の説明を参照されたい。

【００９４】本実施例のディスパッチャのアルゴリズム(図20)は、システム中のいずれかのプロセッサが利用可能になったとき、同プロセッサにより実行される。

【００９５】まず、利用可能になったプロセッサを含むプロセッサグループを指定するスレッドグループ中に実行可能なスレッドが存在するかどうか検査する(2001)。これは、図8のカーネルデータ構造において次のように行う。すなわち、例えば、プロセッサPE2を指定するプロセスを検索するには、プロセス管理テーブル(146)中のすべてのスレッドグループ管理構造体(801～803)のプロセッサグループ指定変数(811～813)を参照し、同変数がプロセッサPE2を指定するすべてのスレッドグループ(801と803で管理されるグループ)のスレッド構造体(611,612,614および、616,617,618)のスレッド実行状態変数(621,622,624および、626,627,628)を参照する。

【００９６】これらのいずれかが「実行可能(READY)」であれば、それらのスレッドの一つを同プロセッサへディスパッチする(2002)。

【００９７】これらのいずれもが「実行可能(READY)」でなければ、システム中のすべてのスレッドのうち、実行可能なものが存在するかどうか検査する(2003)。これ

は、図7のカーネルデータ構造において次のように行う。すなわち、プロセス管理テーブル(146)中のすべてのスレッドグループ管理構造体(801～803)のプロセッサグループ指定変数(811～813)を参照し、同変数がプロセッサPE2を指定しない(PE2を指定するスレッドは検査済みなので。)すべてのスレッドグループ(802で管理されるグループ)のスレッド構造体(613,615)のスレッド実行状態変数(623,625)を参照する。

【００９８】これらのいずれかが「実行可能(READY)」であれば、それらのスレッドの一つを同プロセッサへディスパッチする(2004)。

【００９９】これらのいずれもが「実行可能(READY)」でなければ、当該プロセッサは、プロセッサ管理テーブルのプロセッサ状態変数を「利用可能(idle)」にして、システム中のいずれかのスレッドが実行可能になるのを待つ(2005)。

【０１００】(１１)実施例11

次に、本発明の別の実施例を示す。本実施例の全体図は図1であるが、カーネルデータ構造の詳細が図9に示すデータ構造に置き換わっている。ディスパッチャのアルゴリズムは図21のフローチャートである。

【０１０１】本実施例のカーネルデータ構造(図9)は、図6のプロセッサグループ指定変数(631～638)が、プロセッサ指定変数(931～938)に置き換わった以外は図6と同一である。本図の各プロセッサ指定変数(931～938)は、プロセッサIDを格納することによっていずれかのプロセッサを指定する。

【０１０２】本実施例のディスパッチャのアルゴリズム(図21)は、システム中のいずれかのスレッドが実行可能状態になったとき、システム中のいずれかのプロセッサにより実行される。

【０１０３】まず、実行可能なスレッドが指定するプロセッサが利用可能かどうか検査する(2101)。もし、同プロセッサが利用可能ならば、同プロセッサへディスパッチする(2102)。

【０１０４】もし、同プロセッサが利用可能でないならば、同プロセッサと同一グループ内のいずれかのプロセッサが利用可能かどうか調査する(2103)。(調査の方法は、実施例5を参照のこと。)

もし、同一グループ内のいずれかのプロセッサが利用可能ならば、それらの一つにディスパッチする(2104)。(利用可能なプロセッサが複数あり、それらのうち前回ディスパッチしたプロセッサがあれば、同プロセッサへディスパッチするのが望ましい。)

もし、いずれのプロセッサも利用可能でないならば、システム中のいずれかのプロセッサが利用可能になるのを待つ(2105)。

【０１０５】(１２)実施例12

次に、本発明の別の実施例を示す。本実施例の全体図およびカーネルデータ構造の詳細は、図1と図9であり、実施例11と同一である。ただし、ディスパッチャのアルゴリズムは図22のフローチャートである。

【０１０６】本実施例のディスパッチャのアルゴリズム(図22)は、システム中のいずれかのプロセッサが利用可能になったとき、同プロセッサにより実行される。

【０１０７】まず、利用可能になったプロセッサを指定する実行可能なスレッドが存在するかどうか検査する(2201)。(検査方法は実施例6と同様。)

これらスレッドのいずれかが実行可能(READY)であれば、それらのスレッドの一つを同プロセッサへディスパッチする(2202)。(条件に該当するスレッドが複数の場合には前回当該プロセッサへディスパッチしたスレッドをへディスパッチするのが望ましい。)

これらのいずれもが実行可能でなければ、当該利用可能なプロセッサと同一グループに属するプロセッサを含むプロセッサグループを指定する実行可能なスレッドが存在するか検査する。

【０１０８】いずれかが実行可能(READY)であれば、それらのスレッドの一つを同プロセッサへディスパッチする(2204)。(条件に該当するスレッドが複数の場合には前回当該プロセッサへディスパッチしたスレッドをへディスパッチするのが望ましい。)

これらのいずれもが「実行可能(READY)」でなければ、当該プロセッサは、プロセッサ管理テーブルのプロセッサ状態変数を「利用可能(idle)」にして、システム中のいずれかのスレッドが実行可能になるのを待つ(2205)。

【０１０９】(１３)実施例13

次に、本発明の別の実施例を示す。本実施例の全体図は図1であるが、カーネルデータ構造の詳細が図10に示すデータ構造に置き換わっている。ディスパッチャのアルゴリズムは図21のフローチャートである。

【０１１０】本実施例のカーネルデータ構造(図10)は、図7のプロセッサグループ指定変数(701～703)が、プロセッサ指定変数(1001～1003)に置き換わった以外は図7と同一である。本図の各プロセッサ指定変数(1001～1003)は、プロセッサIDを格納することによっていずれかのプロセッサを指定する。

【０１１１】本実施例のディスパッチャのアルゴリズム(図21)は、システム中のいずれかのスレッドが実行可能状態になったとき、システム中のいずれかのプロセッサにより実行される。

【０１１２】まず、実行可能なスレッドが属するプロセスが指定するプロセッサが利用可能かどうか検査する(2101)。もし、同プロセッサが利用可能ならば、同プロセッサへディスパッチする(2102)。

【０１１３】もし、同プロセッサが利用可能でないならば、同プロセッサと同一グループ内のいずれかのプロセッサが利用可能かどうか調査する(2103)。(調査の方法は、実施例5を参照のこと。)

もし、同一グループ内のいずれかのプロセッサが利用可

能ならば、それらの一つにディスパッチする (2104)。（利用可能なプロセッサが複数あり、それらのうち前回ディスパッチしたプロセッサがあれば、同プロセッサへディスパッチするのが望ましい。）
もし、いずれのプロセッサも利用可能でないならば、システム中のいずれかのプロセッサが利用可能になるのを待つ (2105)。
【０１１４】（１４）実施例14
次に、本発明の別の実施例を示す。本実施例の全体図およびカーネルデータ構造の詳細は、図1と図10であり、実施例13と同一である。ただし、ディスパッチャのアルゴリズムは図22のフローチャートである。
【０１１５】本実施例のディスパッチャのアルゴリズム (図22)は、システム中のいずれかのプロセッサが利用可能になったとき、同プロセッサにより実行される。
【０１１６】まず、利用可能になったプロセッサを指定するプロセス内に実行可能なスレッドが存在するかどうか検査する (2201)。（検査方法は実施例7と同様。）
これらスレッドのいずれかが実行可能 (READY)であれば、それらのスレッドの一つを同プロセッサへディスパッチする (2202)。（条件に該当するスレッドが複数の場合には前回当該プロセッサへディスパッチしたスレッドをへディスパッチするのが望ましい。）
これらのいずれもが実行可能でなければ、当該利用可能なプロセッサと同一グループに属するプロセッサを含むプロセッサグループを指定するプロセス内の実行可能なスレッドが存在するか検査する。
【０１１７】いずれかが実行可能 (READY)であれば、それらのスレッドの一つを同プロセッサへディスパッチする (2204)。（条件に該当するスレッドが複数の場合には前回当該プロセッサへディスパッチしたスレッドをへディスパッチするのが望ましい。）
これらのいずれもが「実行可能 (READY)」でなければ、当該プロセッサは、プロセッサ管理テーブルのプロセッサ状態変数を「利用可能 (idle)」にして、システム中のいずれかのスレッドが実行可能になるのを待つ (2205)。
【０１１８】
【発明の効果】本発明によれば、スレッドのプロセッサアフィニティが失敗した場合でも、グループアフィニティを使用することにより、キャッシュを共有しないプロセッサ間でのスレッドの移動頻度が削減され、これによりキャッシュを有効に使用する（ヒット率を向上させる）ことができる。
【０１１９】さらに、本発明によれば、スレッドあるいはプロセス毎にプロセッサグループを指定することが出来るため、例えば、データ・ワーキングセットが一致し、かつ、並列度の高い（複数のプロセッサにより実行した場合、並列に処理される可能性の高い）複数のスレッドを、キャッシュを共有する複数のプロセッサへアフィニティ付け、あるいは、動作が異なり関連性の無いプロセスを互いに異なるプロセッサグループにアフィニティ付けすることが出来る。具体的には、例えば、データベース・システムのように、システムに共通なテーブルを頻繁にアクセスするプロセス・スレッド群と、ディスクI/Oを専門に担当するデーモンプロセスを、システム中の適当なプロセッサへアフィニティ付けすることにより、キャッシュを有効に使用することができる。
【図面の簡単な説明】
【図1】本発明による計算機システムの全体図
【図2】従来の計算機システム（共有キャッシュなし）の全体図
【図3】従来の計算機システム（共有キャッシュあり）の全体図
【図4】本発明による計算機システムの実施例1,2のカーネルデータ構造
【図5】本発明による計算機システムの実施例1,2の別のカーネルデータ構造
【図6】本発明による計算機システムの実施例3,5,6のカーネルデータ構造
【図7】本発明による計算機システムの実施例4,7,8のカーネルデータ構造
【図8】本発明による計算機システムの実施例9,10のカーネルデータ構造
【図9】本発明による計算機システムの実施例11,12のカーネルデータ構造
【図10】本発明による計算機システムの実施例13,14のカーネルデータ構造
【図11】従来の計算機システムのカーネルデータ構造
【図12】本発明による計算機システムの実施例1のディスパッチャのアルゴリズム
【図13】本発明による計算機システムの実施例2のディスパッチャのアルゴリズム
【図14】本発明による計算機システムの実施例3のディスパッチャのアルゴリズム
【図15】本発明による計算機システムの実施例4のディスパッチャのアルゴリズム
【図16】本発明による計算機システムの実施例5,7のディスパッチャのアルゴリズム
【図17】本発明による計算機システムの実施例6のディスパッチャのアルゴリズム
【図18】本発明による計算機システムの実施例8のディスパッチャのアルゴリズム
【図19】本発明による計算機システムの実施例9のディスパッチャのアルゴリズム
【図20】本発明による計算機システムの実施例10のディスパッチャのアルゴリズム
【図21】本発明による計算機システムの実施例11,13のディスパッチャのアルゴリズム
【図22】本発明による計算機システムの実施例12,14のディスパッチャのアルゴリズム

31

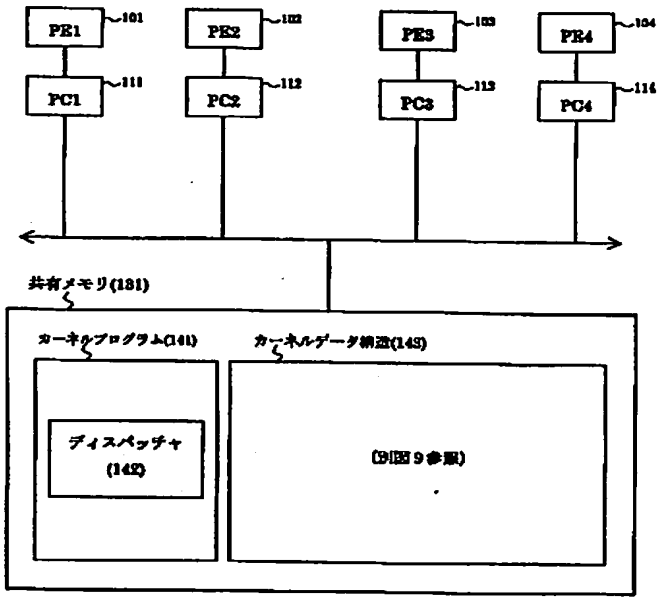
【符号の説明】

101,102,103,104…プロセッサ、111,112,113,114…プライベートキャッシュ、121,122…共有キャッシュ、131…共有メモリ、141…カーネルプログラム、142…ディスパッチャ、143…カーネルデータ構造、144,145…プロセッサグループ管理テーブル、146…プロセス管理テーブル、151,152,153,154…プロセッサ管理テーブル、161,162,163,164…プロセッサ状態変数、171,172,173,174…アフィニティ管理ポインタ、181～188…スレッド構造体、401～408…スレッド実行状態変数、601～603…プロセス構造体、611～618…スレッド構造体、621～628…スレッド実行状態変数、631～638…プロセッサグループ指定変数、701～703…プロセッサグループ指定変数、801～803…スレッドグループ管理構造体、811～813…プロセッサグループ指定変数、821～823…スレッドグループ管理ポインタ、831～838…スレッドグループリンクポインタ、931～938…プロセッサ指定変数、1001～1003…プロセッサグループ指定変数。

32

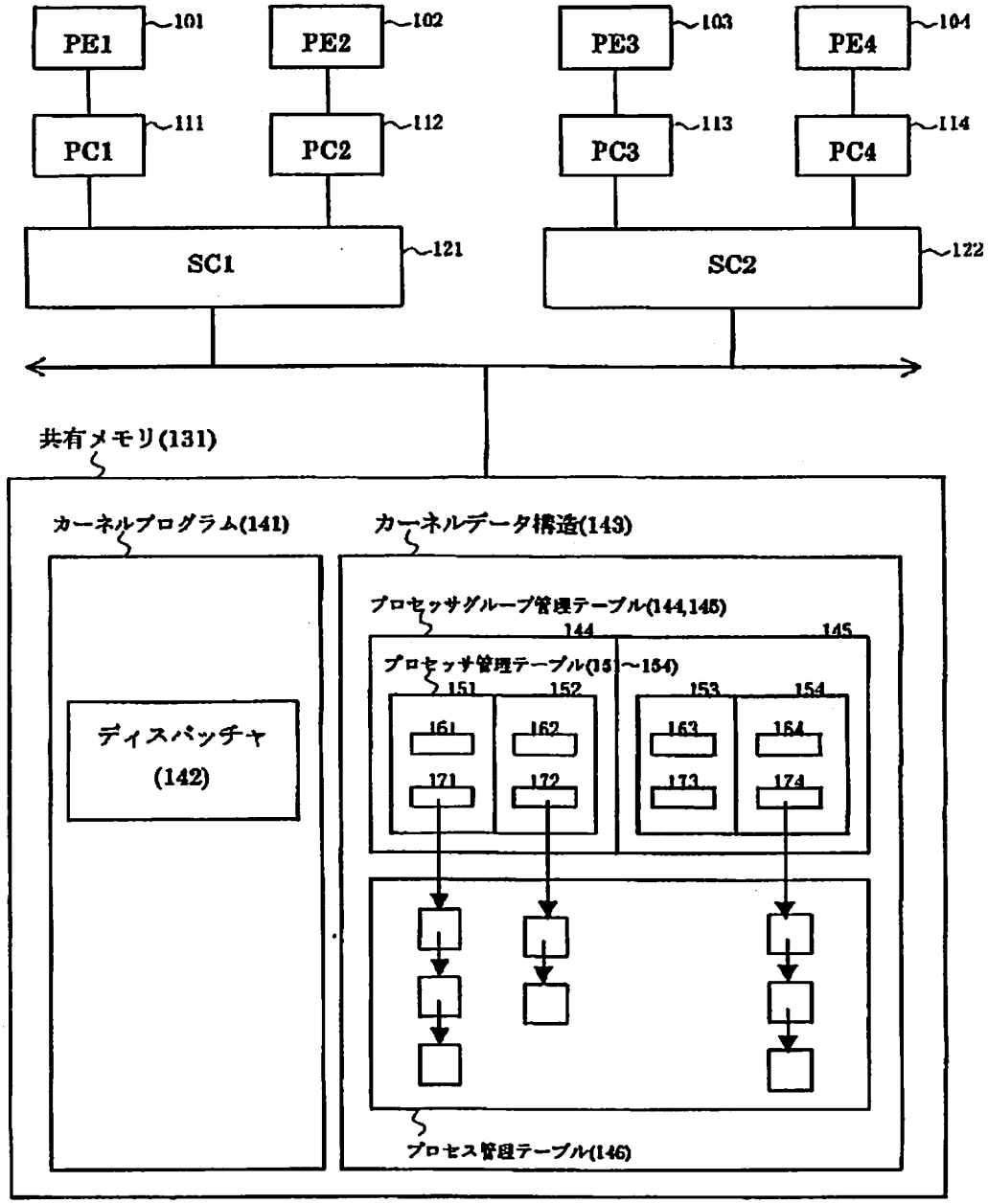
【図2】

図2



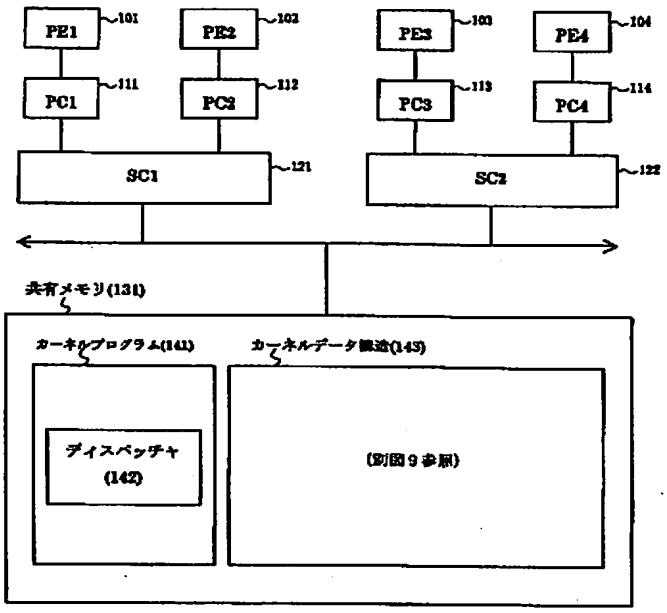
【図1】

図１


PE1
101
PE2
102
PE3
103
PE4
104
PC1
111
PC2
112
PC3
113
PC4
114
SC1
121
SC2
122
共有メモリ(131)
カーネルプログラム(141)
カーネルデータ構造(143)
ディスパッチャ
(142)
プロセッサグループ管理テーブル(144,145)
144
145
プロセッサ管理テーブル(151～154)
151
152
153
154
161
162
163
164
171
172
173
174
プロセス管理テーブル(146)

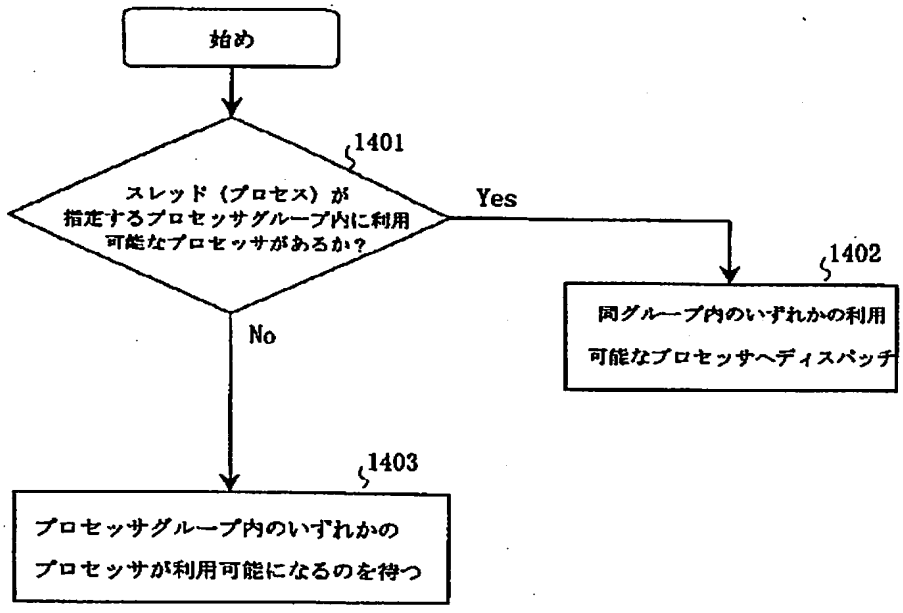
【図３】

図３


PE1
101
PE2
102
PE3
103
PE4
104
PC1
111
PC2
112
PC3
113
PC4
114
SC1
121
SC2
122
共有メモリ(131)
カーネルプログラム(141)
カーネルデータ構造(143)
ディスパッチャ
(142)
(別図９参照)

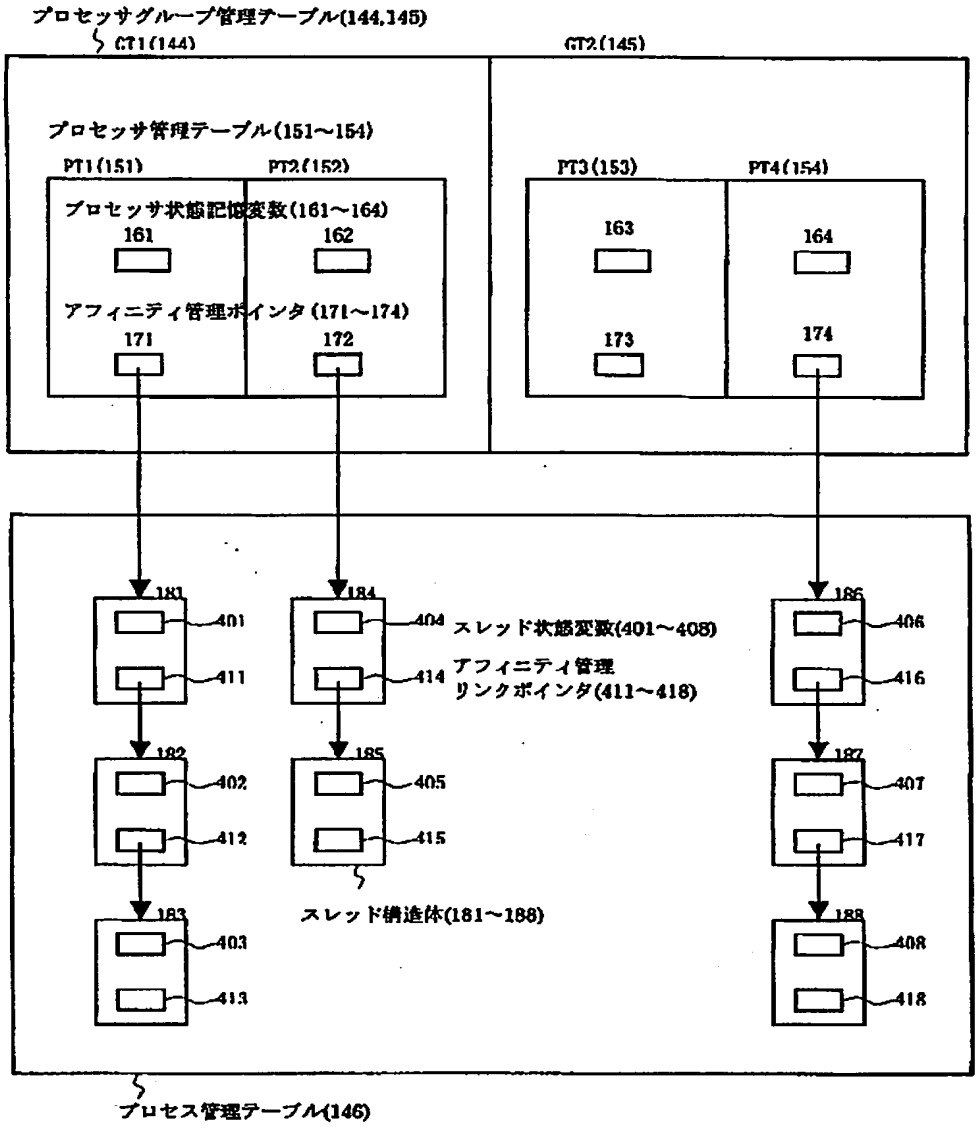
【図１４】

図 14


始め
1401
スレッド（プロセス）が
指定するプロセッサグループ内に利用
可能なプロセッサがあるか？
Yes
No
1402
同グループ内のいずれかの利用
可能なプロセッサへディスパッチ
1403
プロセッサグループ内のいずれかの
プロセッサが利用可能になるのを待つ

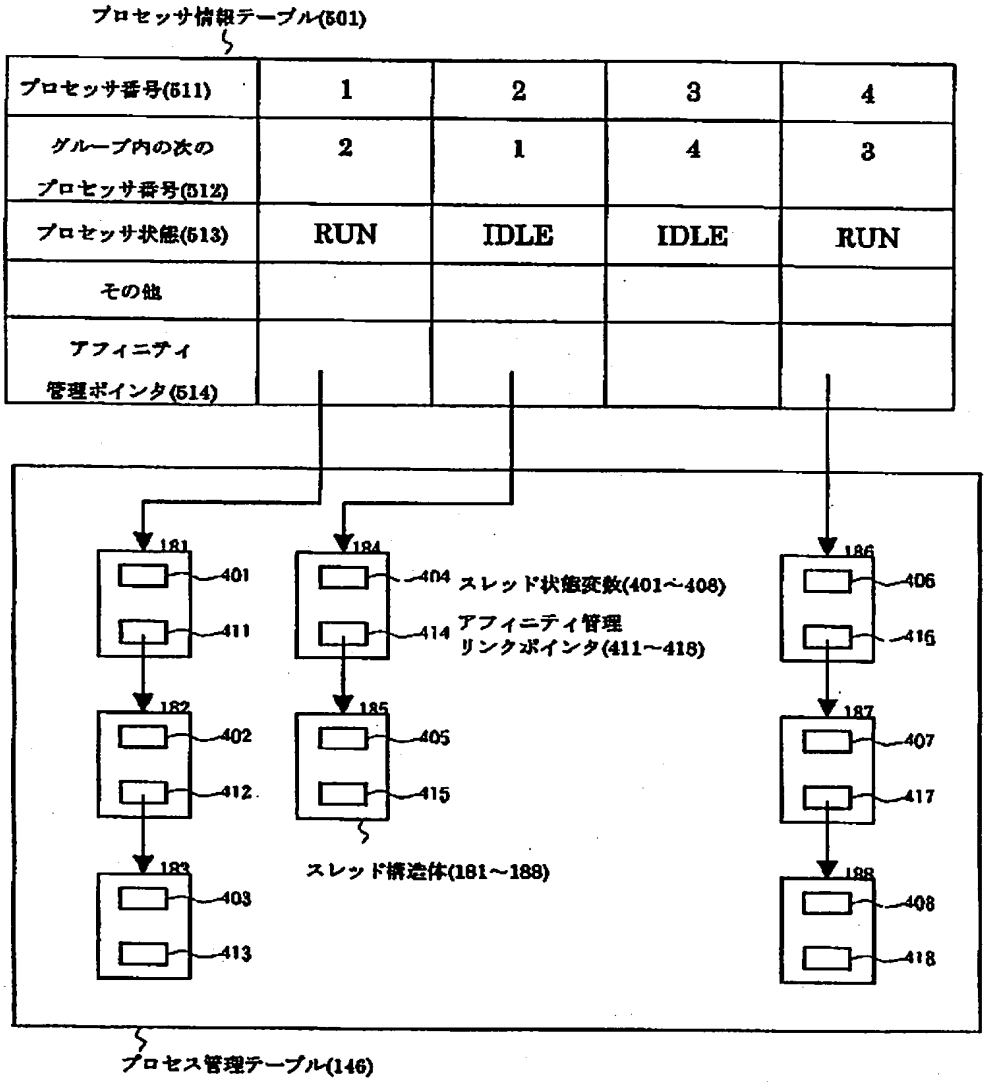
【 図4 】

図４


プロセッサグループ管理テーブル(144,145)
GT1(144)
GT2(145)
プロセッサ管理テーブル(151～154)
PT1(151)
PT2(152)
PT3(153)
PT4(154)
プロセッサ状態記憶変数(161～164)
161
162
163
164
アフィニティ管理ポインタ(171～174)
171
172
173
174
181
182
183
184
185
186
187
188
401
402
403
404
405
406
407
408
411
412
413
414
415
416
417
418
スレッド状態変数(401～408)
アフィニティ管理
リンクポインタ(411～418)
スレッド構造体(181～188)
プロセス管理テーブル(146)

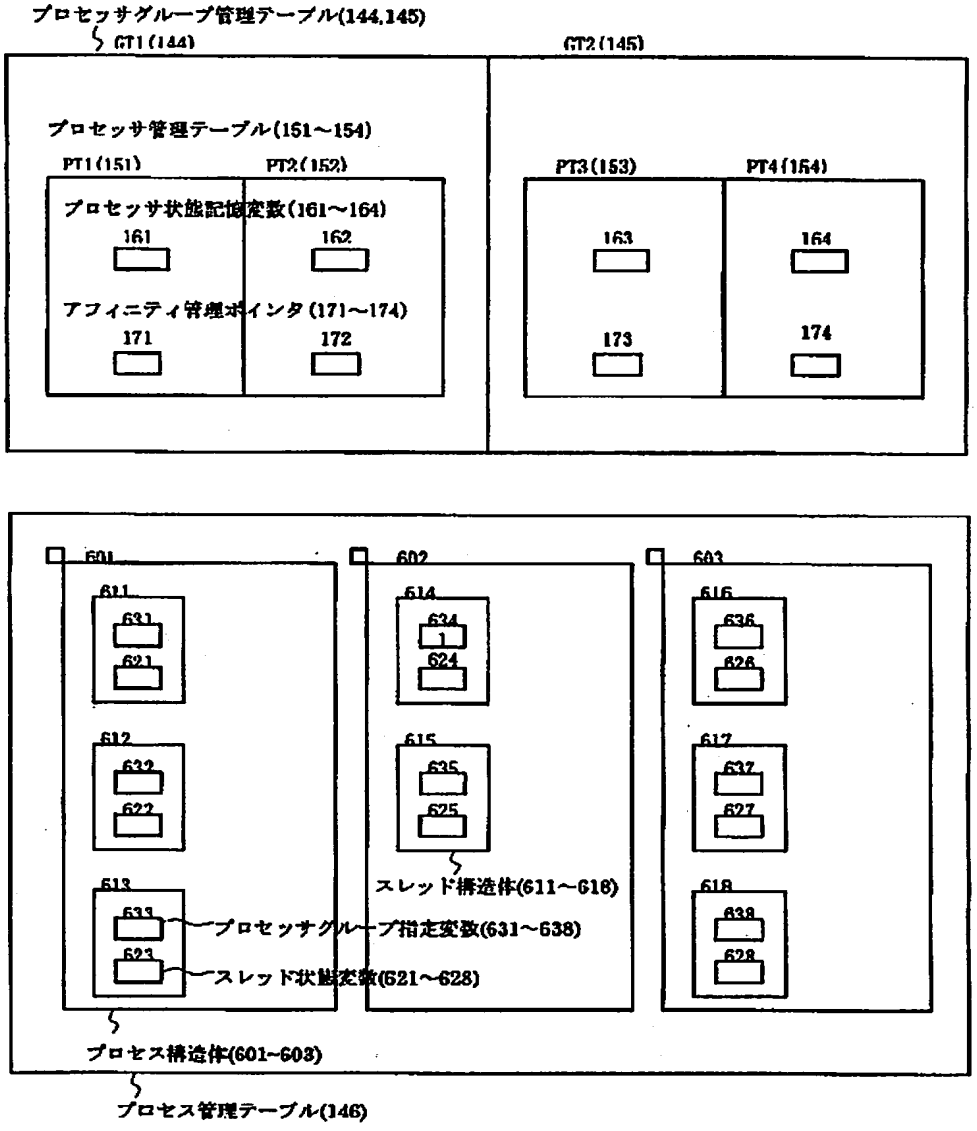
【 図5 】

図５


プロセッサ情報テーブル(501)
プロセッサ番号(511)
1
2
3
4
グループ内の次のプロセッサ番号(512)
2
1
4
3
プロセッサ状態(513)
RUN
IDLE
IDLE
RUN
その他
アフィニティ管理ポインタ(514)
181
401
411
182
402
412
183
403
413
184
404
414
185
405
415
186
406
416
187
407
417
188
408
418
スレッド状態変数(401～408)
アフィニティ管理リンクポインタ(411～418)
スレッド構造体(181～188)
プロセス管理テーブル(146)

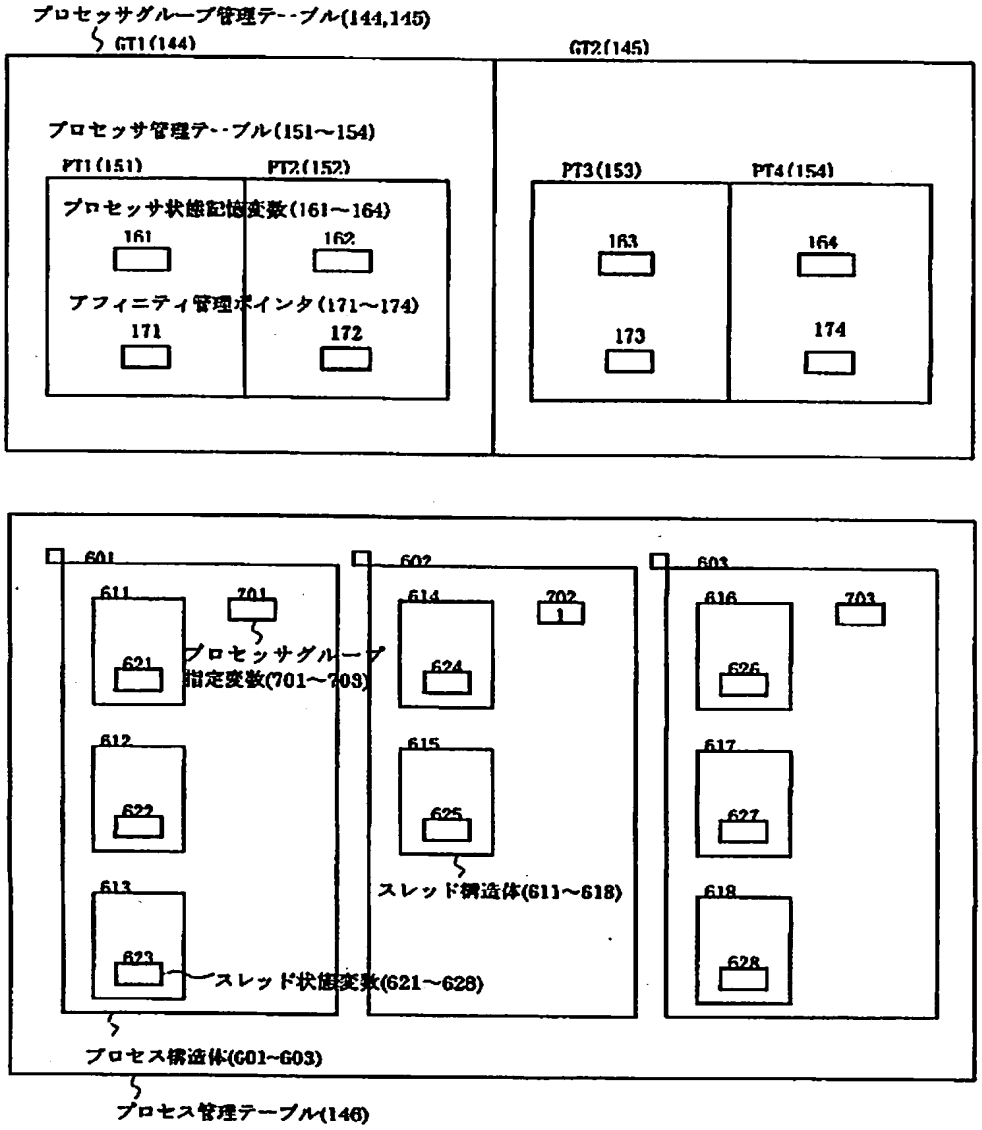
【 図6 】

図６


プロセッサグループ管理テーブル(144,145)
GT1(144)
GT2(145)
プロセッサ管理テーブル(151～154)
PT1(151)
PT2(152)
PT3(153)
PT4(154)
プロセッサ状態記憶変数(161～164)
161
162
163
164
アフィニティ管理ポインタ(171～174)
171
172
173
174
601
602
603
611
631
621
612
632
622
613
633
623
614
634
624
615
635
625
616
636
626
617
637
627
618
638
628
スレッド構造体(611～618)
プロセッサグループ指定変数(631～638)
スレッド状態変数(621～628)
プロセス構造体(601～603)
プロセス管理テーブル(146)

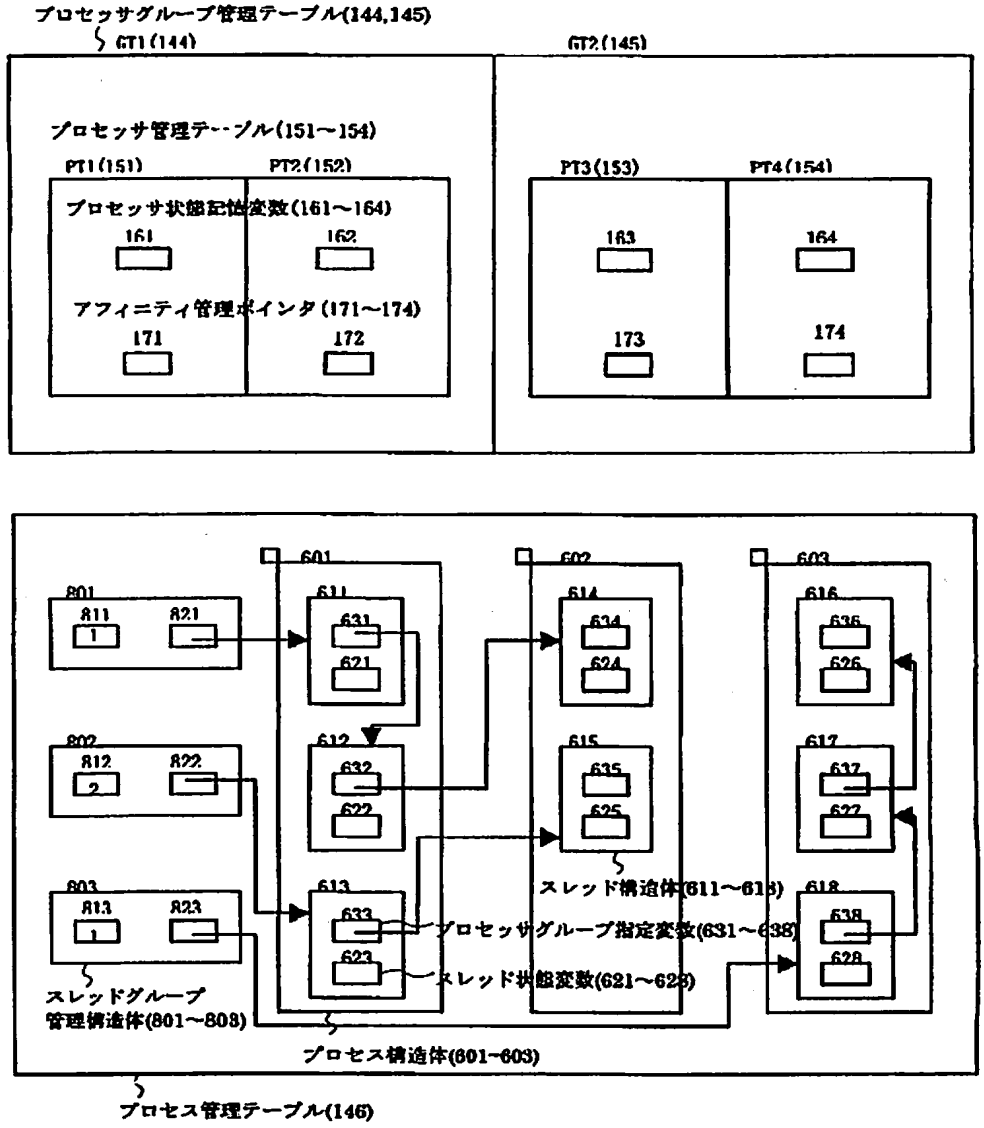
【 図7 】

図7


プロセッサグループ管理テーブル(144,145)
GT1(144)
GT2(145)
プロセッサ管理テーブル(151～154)
PT1(151)
PT2(152)
PT3(153)
PT4(154)
プロセッサ状態記憶変数(161～164)
161
162
163
164
アフィニティ管理ポインタ(171～174)
171
172
173
174
601
602
603
611
701
614
702
616
703
621
プロセッサグループ
指定変数(701～703)
624
626
612
615
617
622
625
627
613
618
623
628
スレッド構造体(611～618)
スレッド状態変数(621～628)
プロセス構造体(601～603)
プロセス管理テーブル(146)

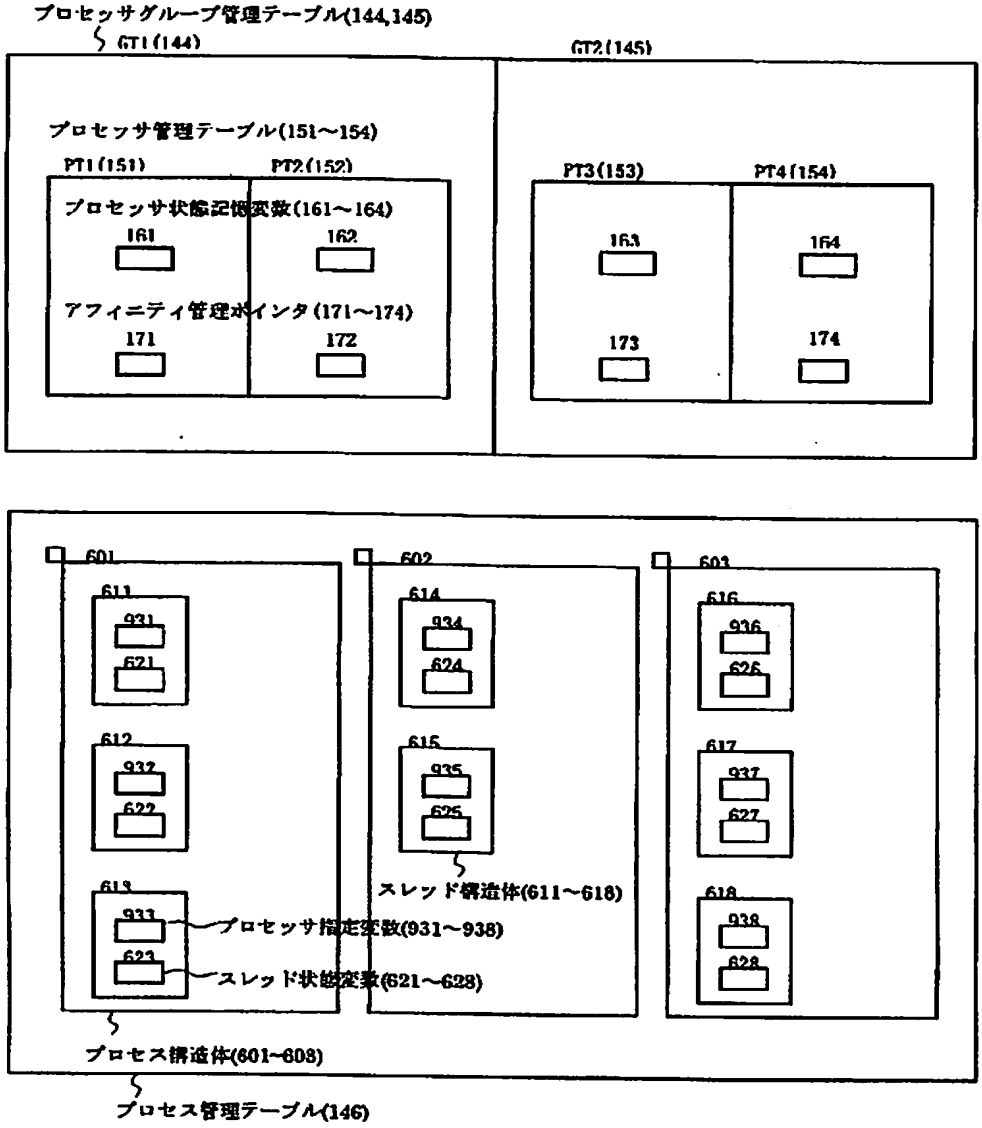
【 図8 】

図８


プロセッサグループ管理テーブル(144,145)
GT1(144)
GT2(145)
プロセッサ管理テーブル(151～154)
PT1(151)
PT2(152)
PT3(153)
PT4(154)
プロセッサ状態記憶変数(161～164)
161
162
163
164
アフィニティ管理ポインタ(171～174)
171
172
173
174
601
602
603
801
802
803
811
821
812
822
813
823
611
612
613
614
615
616
617
618
621
622
623
624
625
626
627
628
631
632
633
634
635
636
637
638
スレッド構造体(611～618)
プロセッサグループ指定変数(631～638)
スレッド状態変数(621～628)
スレッドグループ
管理構造体(801～808)
プロセス構造体(601～603)
プロセス管理テーブル(146)

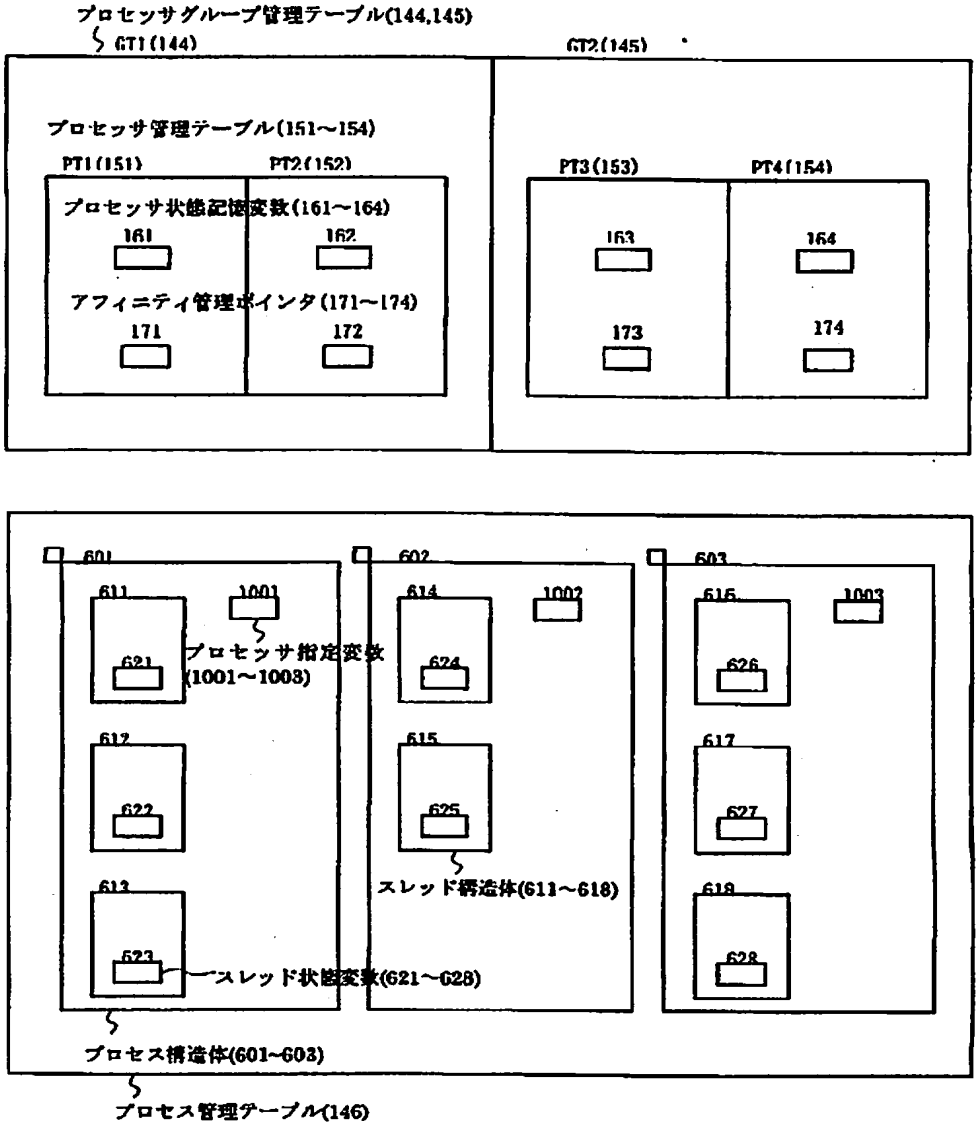
【 図9 】

図 9


プロセッサグループ管理テーブル(144,145)
GT1(144)
GT2(145)
プロセッサ管理テーブル(151～154)
PT1(151)
PT2(152)
PT3(153)
PT4(154)
プロセッサ状態記憶変数(161～164)
161
162
163
164
アフィニティ管理ポインタ(171～174)
171
172
173
174
601
602
603
611
931
621
612
932
622
613
933
623
614
934
624
615
935
625
616
936
626
617
937
627
618
938
628
スレッド構造体(611～618)
プロセッサ指定変数(931～938)
スレッド状態変数(621～628)
プロセス構造体(601～603)
プロセス管理テーブル(146)

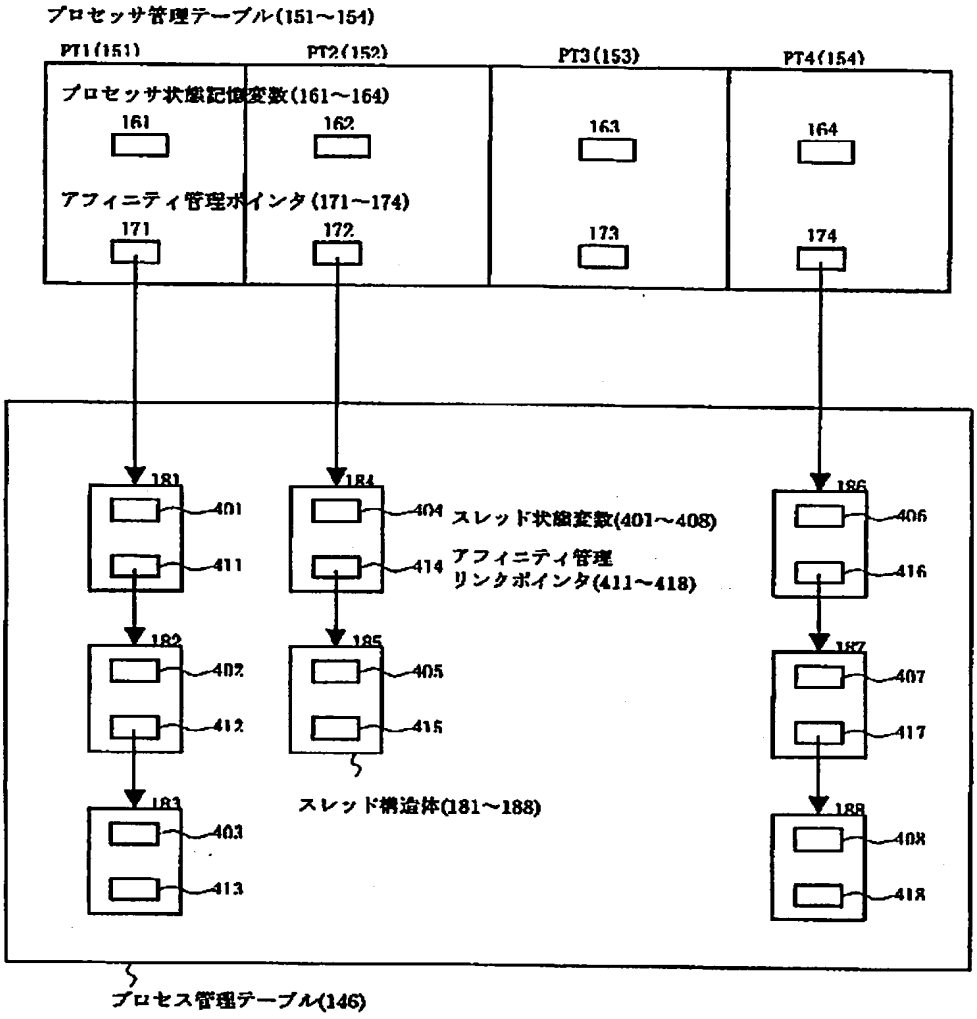
【図１０】

図 10


プロセッサグループ管理テーブル(144,145)
GT1(144)
GT2(145)
プロセッサ管理テーブル(151～154)
PT1(151)
PT2(152)
PT3(153)
PT4(154)
プロセッサ状態記憶変数(161～164)
161
162
163
164
アフィニティ管理ポインタ(171～174)
171
172
173
174
601
602
603
611
1001
614
1002
616
1003
621
プロセッサ指定変数
(1001～1003)
624
626
612
615
617
622
625
627
613
618
623
628
スレッド構造体(611～618)
スレッド状態変数(621～628)
プロセス構造体(601～603)
プロセス管理テーブル(146)

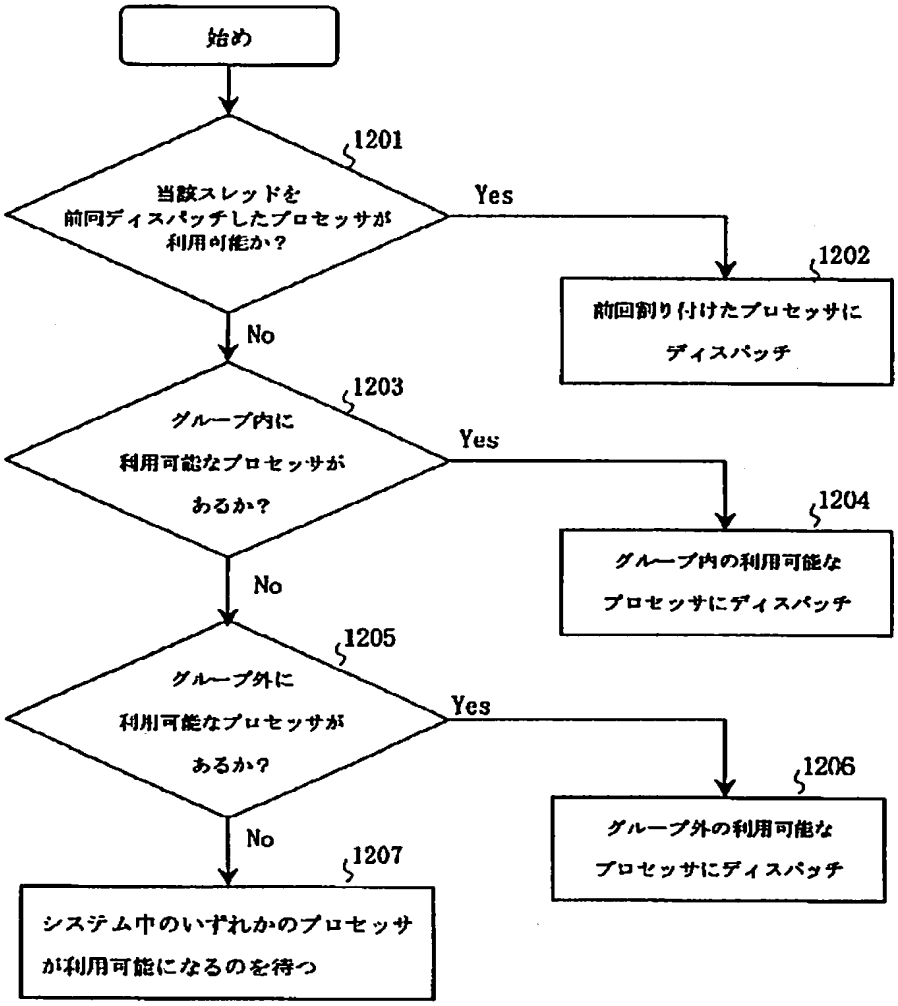
【 図1 1 】

図 11


プロセッサ管理テーブル(151～154)
PT1(151)
PT2(152)
PT3(153)
PT4(154)
プロセッサ状態記憶変数(161～164)
161
162
163
164
アフィニティ管理ポインタ(171～174)
171
172
173
174
181
182
183
184
185
186
187
188
401
402
403
404
405
406
407
408
411
412
413
414
415
416
417
418
スレッド状態変数(401～408)
アフィニティ管理
リンクポインタ(411～418)
スレッド構造体(181～188)
プロセス管理テーブル(146)

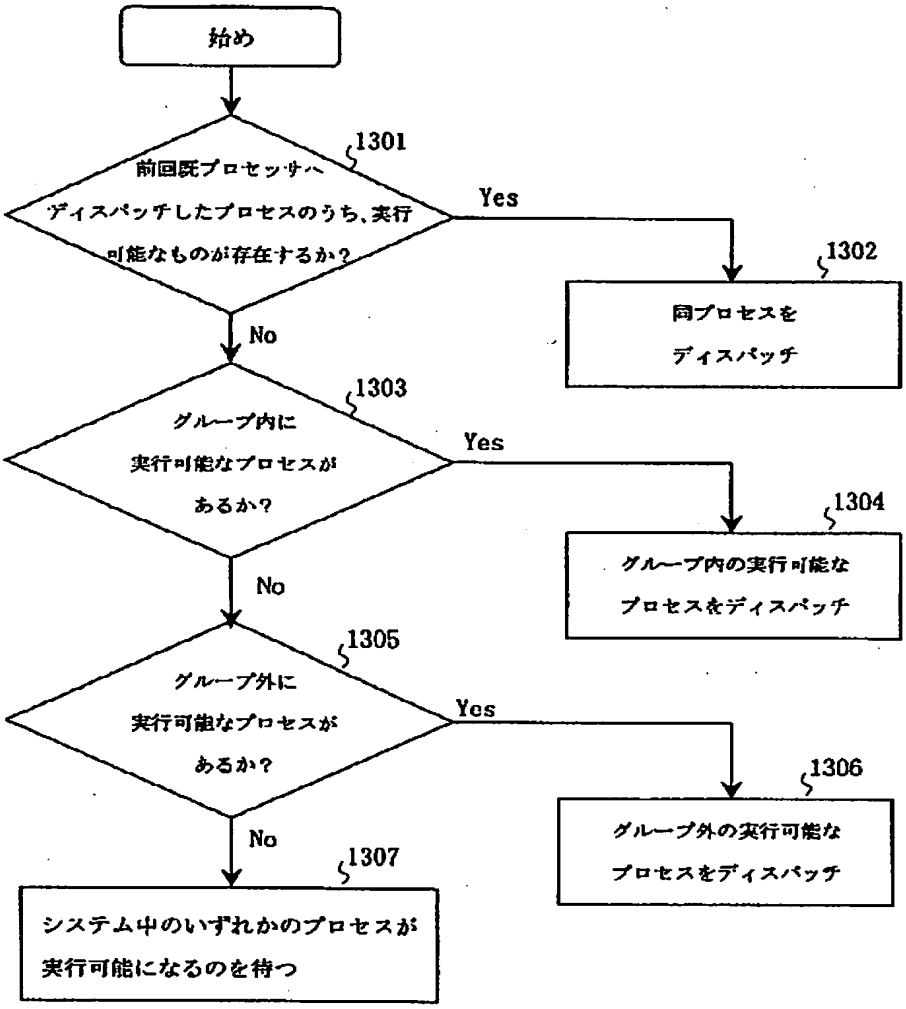
【図１２】

図 12


始め
1201
当該スレッドを
前回ディスパッチしたプロセッサが
利用可能か？
Yes
1202
前回割り付けたプロセッサに
ディスパッチ
No
1203
グループ内に
利用可能なプロセッサが
あるか？
Yes
1204
グループ内の利用可能な
プロセッサにディスパッチ
No
1205
グループ外に
利用可能なプロセッサが
あるか？
Yes
1206
グループ外の利用可能な
プロセッサにディスパッチ
No
1207
システム中のいずれかのプロセッサ
が利用可能になるのを待つ

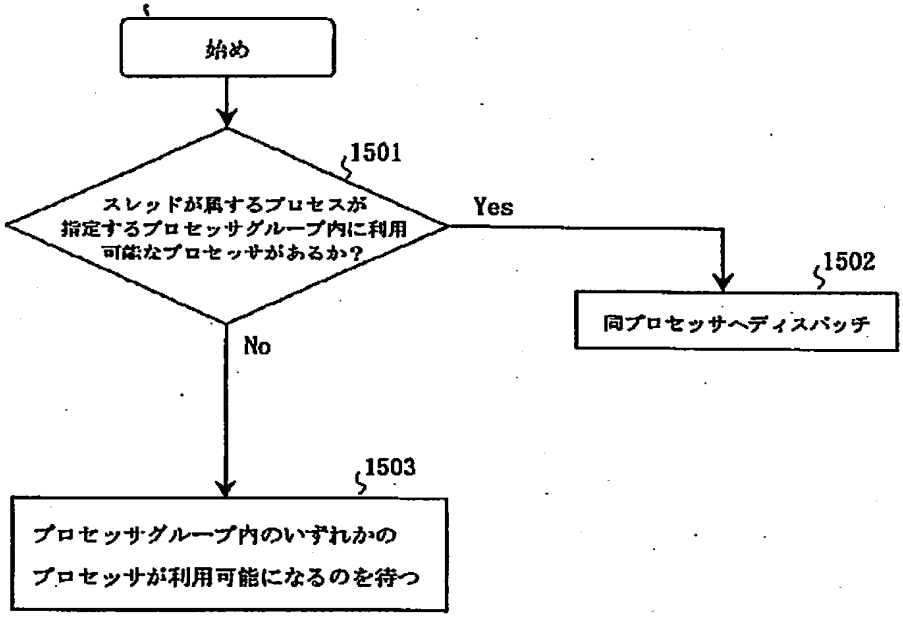
【 図1３ 】

図 13


始め
1301
前回既プロセッサへディスパッチしたプロセスのうち、実行可能なものが存在するか？
Yes
1302
同プロセスをディスパッチ
No
1303
グループ内に実行可能なプロセスがあるか？
Yes
1304
グループ内の実行可能なプロセスをディスパッチ
No
1305
グループ外に実行可能なプロセスがあるか？
Yes
1306
グループ外の実行可能なプロセスをディスパッチ
No
1307
システム中のいずれかのプロセスが実行可能になるのを待つ

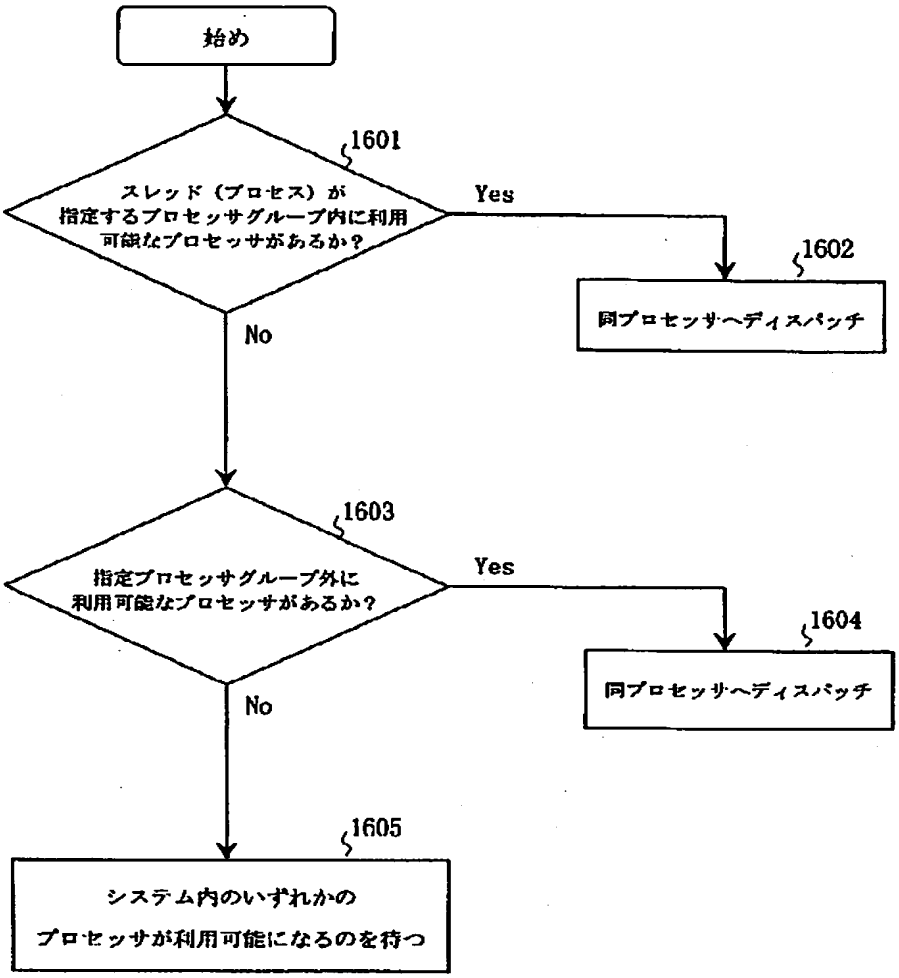
【 図1５ 】

図 15


始め
1501
スレッドが属するプロセスが
指定するプロセッサグループ内に利用
可能なプロセッサがあるか？
Yes
1502
同プロセッサへディスパッチ
No
1503
プロセッサグループ内のいずれかの
プロセッサが利用可能になるのを待つ

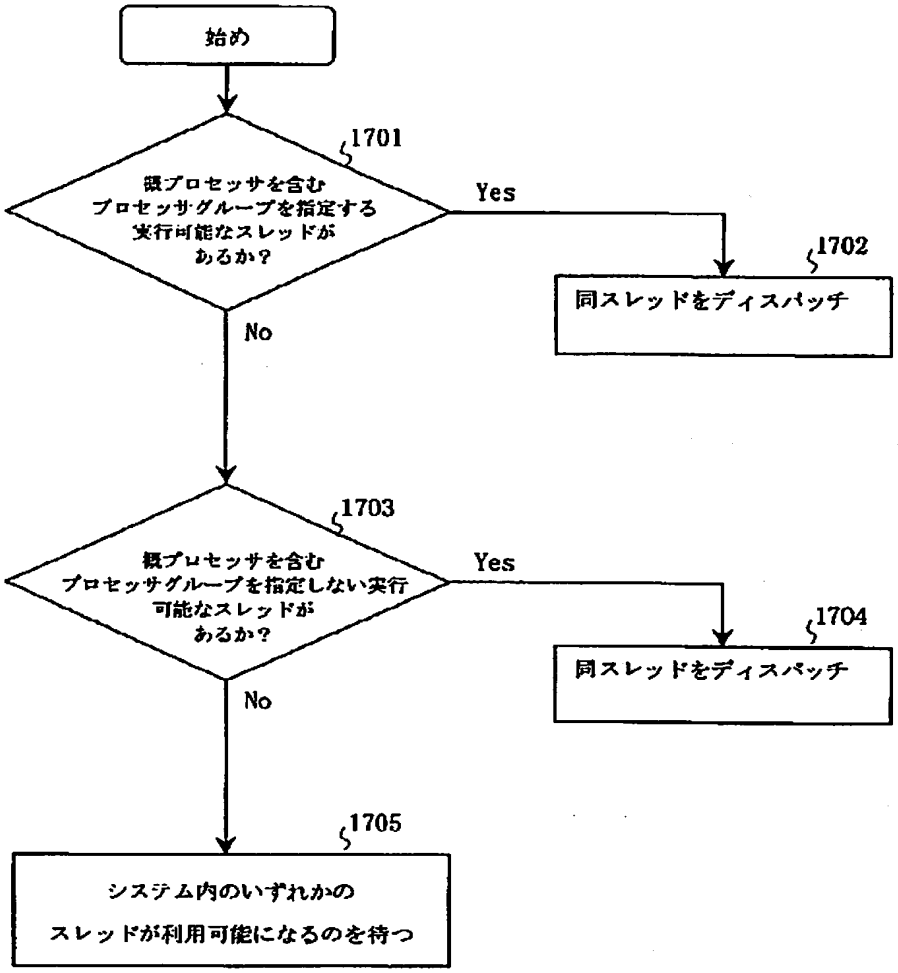
【図１６】

図 16


始め
1601
スレッド（プロセス）が
指定するプロセッサグループ内に利用
可能なプロセッサがあるか？
Yes
1602
同プロセッサへディスパッチ
No
1603
指定プロセッサグループ外に
利用可能なプロセッサがあるか？
Yes
1604
同プロセッサへディスパッチ
No
1605
システム内のいずれかの
プロセッサが利用可能になるのを待つ

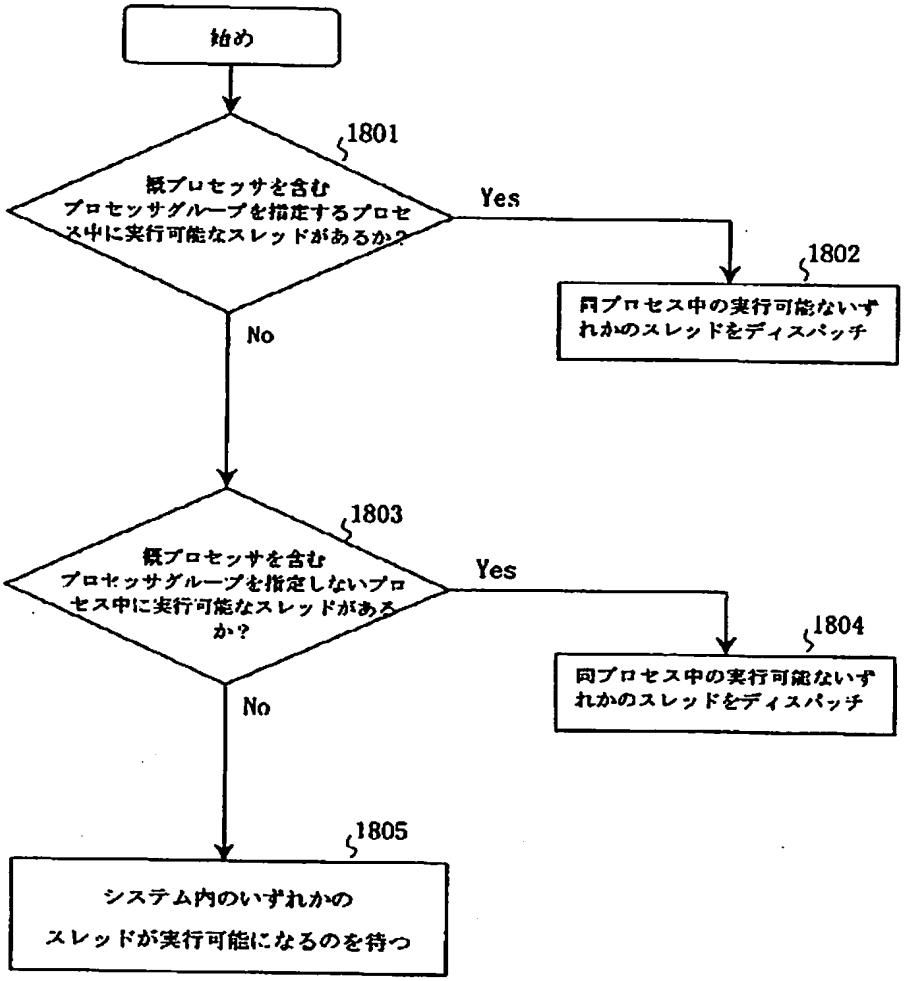
【 図1 7 】

図 17


始め
1701
親プロセッサを含む
プロセッサグループを指定する
実行可能なスレッドが
あるか？
Yes
1702
同スレッドをディスパッチ
No
1703
親プロセッサを含む
プロセッサグループを指定しない実行
可能なスレッドが
あるか？
Yes
1704
同スレッドをディスパッチ
No
1705
システム内のいずれかの
スレッドが利用可能になるのを待つ

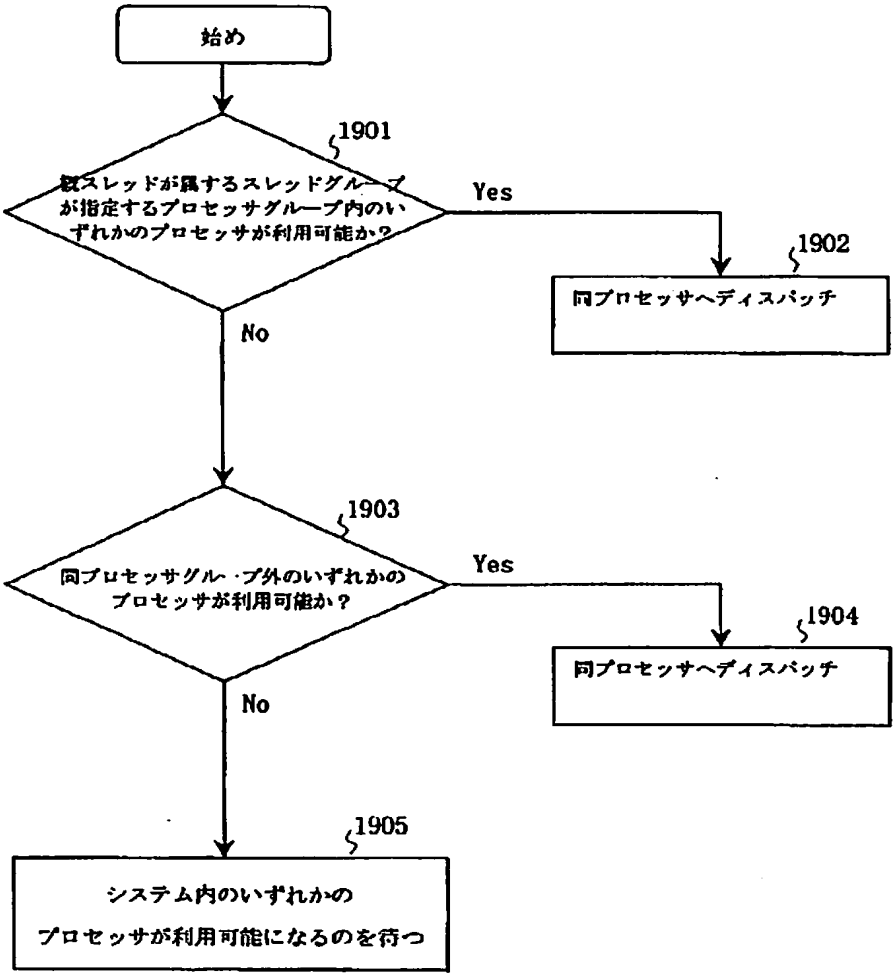
【 図1８ 】

図 18


始め
1801
該プロセッサを含むプロセッサグループを指定するプロセス中に実行可能なスレッドがあるか？
Yes
1802
同プロセス中の実行可能ないずれかのスレッドをディスパッチ
No
1803
該プロセッサを含むプロセッサグループを指定しないプロセス中に実行可能なスレッドがあるか？
Yes
1804
同プロセス中の実行可能ないずれかのスレッドをディスパッチ
No
1805
システム内のいずれかのスレッドが実行可能になるのを待つ

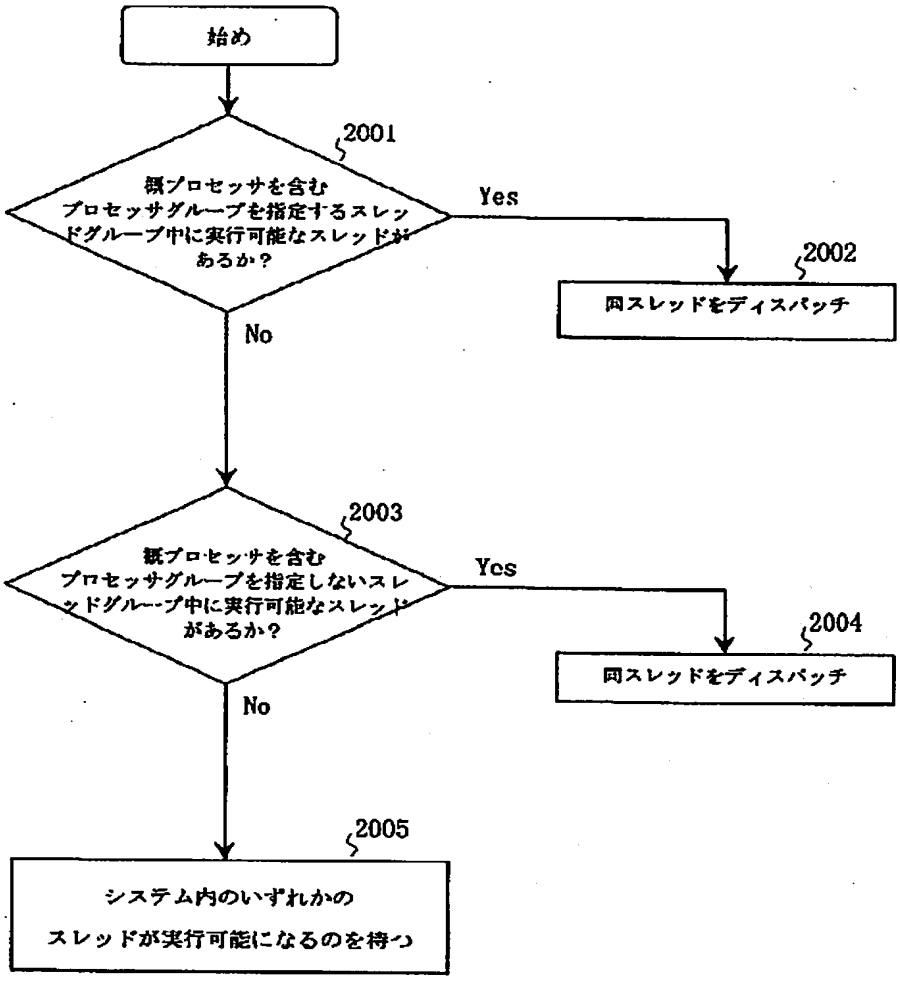
【図１９】

図 19


始め
1901
被スレッドが属するスレッドグループが指定するプロセッサグループ内のいずれかのプロセッサが利用可能か？
Yes
1902
同プロセッサへディスパッチ
No
1903
同プロセッサグループ外のいずれかのプロセッサが利用可能か？
Yes
1904
同プロセッサへディスパッチ
No
1905
システム内のいずれかのプロセッサが利用可能になるのを待つ

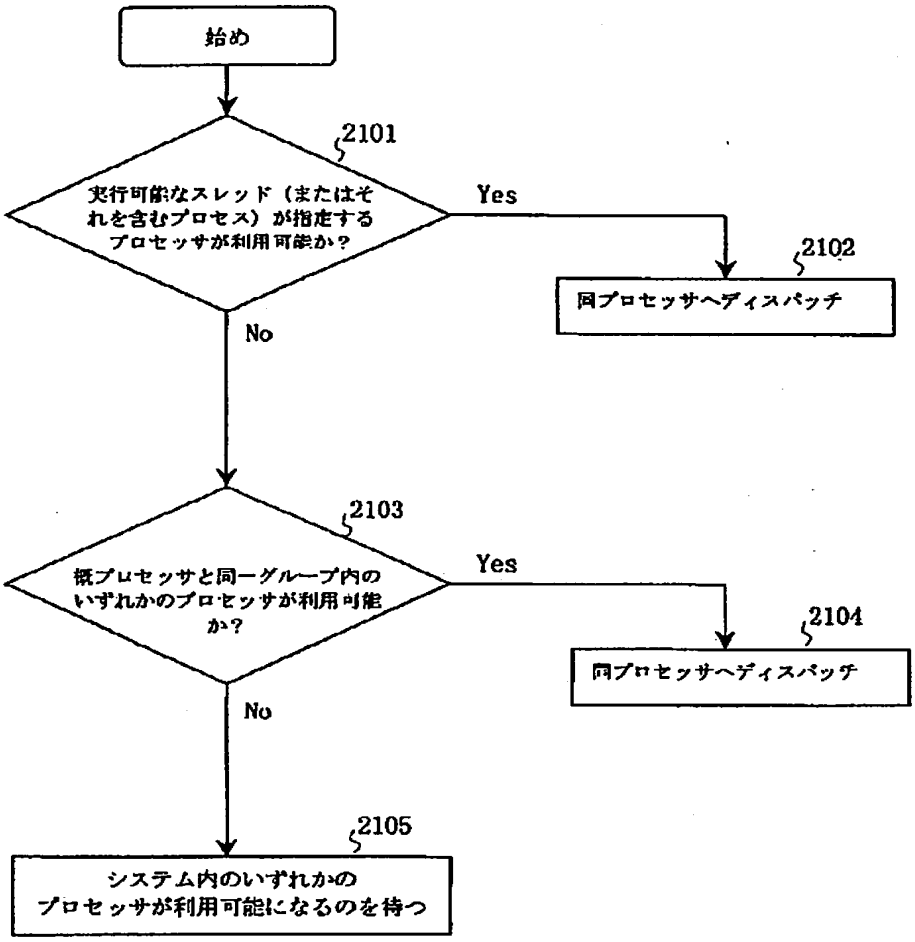
【 図２０ 】

図 20


始め
2001
該プロセッサを含む
プロセッサグループを指定するスレッドグループ中に実行可能なスレッドがあるか？
Yes
2002
同スレッドをディスパッチ
No
2003
該プロセッサを含む
プロセッサグループを指定しないスレッドグループ中に実行可能なスレッドがあるか？
Yes
2004
同スレッドをディスパッチ
No
2005
システム内のいずれかの
スレッドが実行可能になるのを待つ

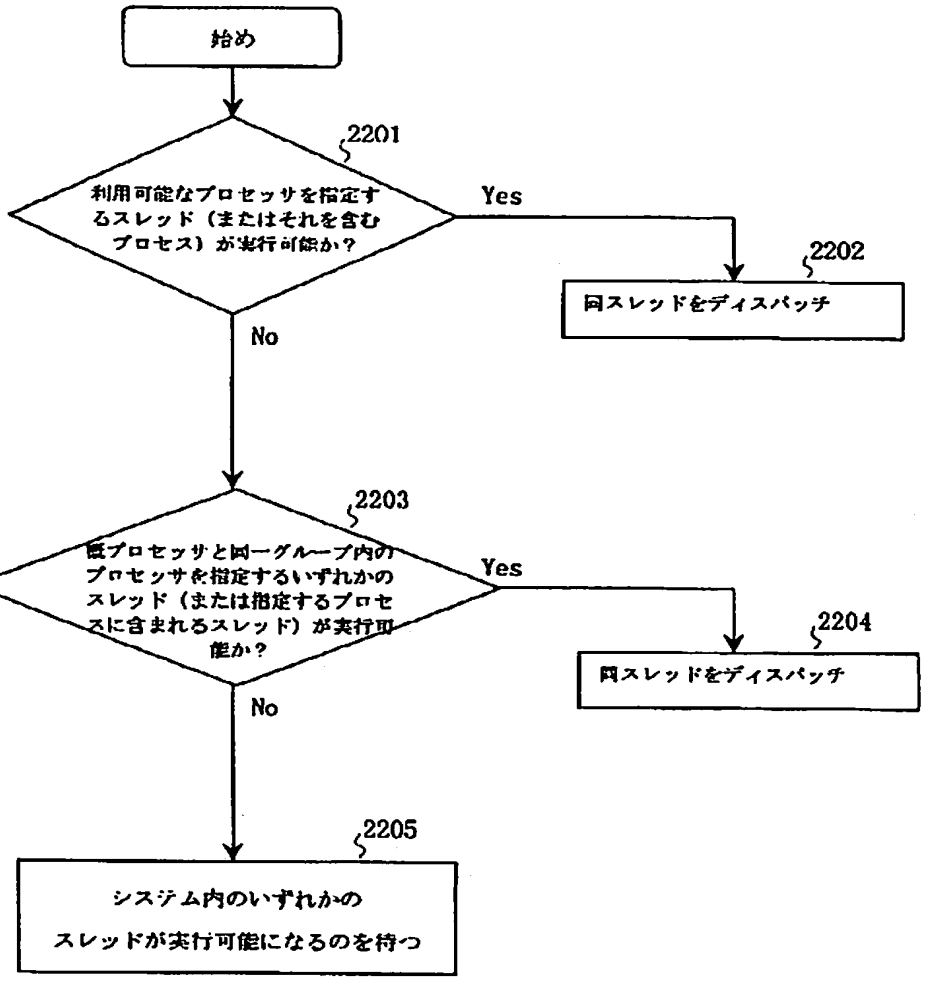
【 図2 1 】

図 21


始め
2101
実行可能なスレッド（またはそれを含むプロセス）が指定するプロセッサが利用可能か？
Yes
2102
同プロセッサへディスパッチ
No
2103
旧プロセッサと同一グループ内のいずれかのプロセッサが利用可能か？
Yes
2104
同プロセッサへディスパッチ
No
2105
システム内のいずれかのプロセッサが利用可能になるのを待つ

【 図2２ 】

図 22



フロントページの続き

(72)発明者　梅野　英典
神奈川県海老名市下今泉810番地　株式会社日立製作所オフィスシステム事業部内